Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/JP05/000451

International filing date: 11 January 2005 (11.01.2005)

Document type: Certified copy of priority document

Document details: Country/Office: JP
Number: 2004-008327
Filing date: 15 January 2004 (15.01.2004)

Date of receipt at the International Bureau: 24 February 2005 (24.02.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

PCT/JP2005/000451

11.01.2005

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

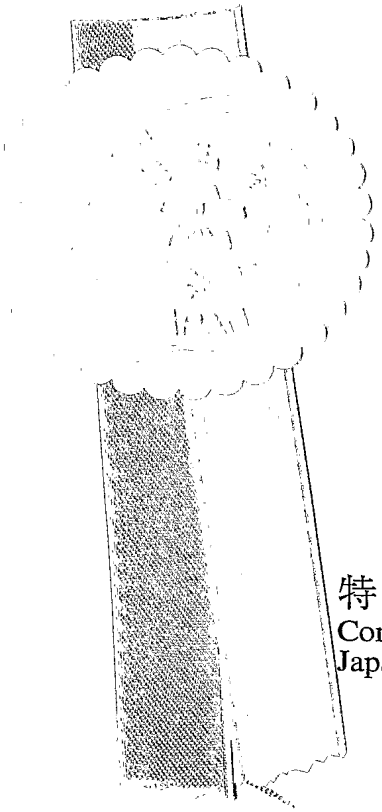
This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００４年　１月１５日

出願番号
Application Number: 特願２００４－００８３２７

[ST. 10/C]: ［ＪＰ２００４－００８３２７］

出願人
Applicant(s): 松下電器産業株式会社



２００５年　２月１４日

特許庁長官
Commissioner,
Japan Patent Office

小川洋

出証番号　出証特２００５－３００９７４１

【書類名】　特許願
【整理番号】　2038250005
【提出日】　平成16年 1月15日
【あて先】　特許庁長官殿
【国際特許分類】　G06F 1/00 340
【発明者】
　【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内
　【氏名】　渕上　竜司
【特許出願人】
　【識別番号】　000005821
　【氏名又は名称】　松下電器産業株式会社
【代理人】
　【識別番号】　100097179
　【弁理士】
　【氏名又は名称】　平野　一幸
【手数料の表示】
　【予納台帳番号】　058698
　【納付金額】　21,000円
【提出物件の目録】
　【物件名】　特許請求の範囲　1
　【物件名】　明細書　1
　【物件名】　図面　1
　【物件名】　要約書　1
　【包括委任状番号】　0013529

【書類名】特許請求の範囲
【請求項１】
メモリのアクセス違反を検出しながら通常の処理を行うステップと、アクセス違反が検出されたら、アドレスに関連付けられた範囲情報と障害復帰情報とに基づいて復帰が可能か否かを判定するステップと、
　復帰が可能なら障害復帰を行うステップとを含む情報処理方法。
【請求項２】
復帰が可能でないなら停止処理を行うステップをさらに含む請求項１記載の情報処理方法。
【請求項３】
障害復帰情報は、固定サイズ属性と可変サイズ属性とを有する請求項１から２記載の情報処理方法。
【請求項４】
可変サイズ属性は、上方伸長属性と下方伸長属性とを有する請求項３記載の情報処理方法。
【請求項５】
前記停止処理を行うステップにおいて、保存可能なデータがあれば、このデータを保存する請求項２記載の情報処理方法。
【請求項６】
前記障害復帰を行うステップにおいて、読み／書きのアクセス種別を判定し、判定結果に基づいて異なる内容の障害復帰を行う請求項１から５記載の情報処理方法。
【請求項７】
アクセス種別が「書き」であるとき、そのまま障害復帰し、アクセス種別が「読み」であるとき読み出し先に所定値を格納して障害復帰する請求項６記載の情報処理方法。
【請求項８】
前記判定するステップにおいて、上方伸長属性があり、かつ、アクセス違反が下方である場合、復帰は可能でないと判定する請求項４記載の情報処理方法。
【請求項９】
前記判定するステップにおいて、下方伸長属性があり、かつ、アクセス違反が上方である場合、復帰は可能でないと判定する請求項４又は８記載の情報処理方法。
【請求項１０】
前記判定するステップにおいて、固定サイズ属性がある場合、復帰は可能でないと判定する請求項４、８又は９記載の情報処理方法。
【請求項１１】
前記障害復帰を行うステップにおいて、アクセス違反した領域とは別の領域を確保し、確保した別の領域にアクセスする請求項１から１０記載の情報処理方法。
【請求項１２】
障害復帰情報は、ターミネータ属性を有し、ターミネータ属性がデータの終端にターミネート値を有することを示す場合、前記障害復帰を行うステップにおいて、データの末尾にターミネート値を付加して障害復帰する請求項１から１１記載の情報処理方法。
【請求項１３】
メモリと、
　アドレス、範囲情報及び障害復帰情報が格納されるアドレスレジスタと、
　前記アドレスレジスタを参照して前記メモリのアクセス違反を検出するアクセス違反検出部と、
　前記メモリをアクセスし通常の処理を行う通常処理部と、
　前記違反検出部がアクセス違反を検出したら前記アドレスレジスタを参照して復帰が可能か否かを判定する障害復帰判定部と、
　前記障害復帰判定部が復帰が可能と判定すると、障害復帰を行う復帰処理部とを備える情報処理装置。

【請求項１４】
前記障害復帰判定部が復帰が可能でないと判定すると、停止処理を行う停止処理部をさらに備える請求項１３記載の情報処理装置。
【請求項１５】
請求項１３又は１４記載の情報処理装置と、前記情報処理装置と障害復帰情報を通信する情報センターとを備える障害情報監視システム。
【請求項１６】
アドレス、範囲情報及び障害復帰情報が格納されるアドレスレジスタと、
　前記アドレスレジスタを参照して外部のメモリをアクセスする命令処理部と、前記アドレスレジスタを参照し前記命令処理部が前記メモリに対しアクセス違反したことを検出すると前記命令処理部に例外信号を出力するアクセス違反検出部とを備えるプロセッサ。
【請求項１７】
ポインタを扱えるプログラム言語で記述されたソースコードをコンパイルする方法であって、
　変数領域を指すポインタ値に、変数領域のアドレスと変数領域の範囲情報とアクセス違反が発生した際の障害復帰情報とを設定するステップを含むコンパイル方法。
【請求項１８】
ポインタを扱えるプログラム言語で記述されたソースコードをコンパイルする方法であって、
　変数領域を指すポインタ値に、変数領域のアドレスと変数領域の範囲情報とを設定するステップと、
　このポインタ値により指される変数領域に、アクセス違反が発生した際の障害復帰情報を設定するステップとを含むコンパイル方法。
【請求項１９】
障害復帰情報は、ソースコードとは別個に管理される請求項１７から１８記載のコンパイル情報。
【請求項２０】
ソースコードに障害復帰情報を付加した中間コードを生成するステップを含む請求項１７から１９記載のコンパイル情報。
【請求項２１】
ポインタを扱えるプログラム言語で記述されたソースコードをコンパイルする言語解析部と、
　前記言語解析部が出力するコードに、変数に対応する障害復帰情報のコードを追加するコード生成部とを備えるコンパイル装置。

【書類名】明細書
【発明の名称】情報処理方法、その装置、それを用いた障害情報監視システム、プロセッサ、コンパイル方法及びその装置
【技術分野】
【０００１】
本発明は、障害復帰機能を強化した情報処理装置及びその関連技術に関するものである。
【背景技術】
【０００２】
近年、組み込み機器向けのプログラムを、Ｃ言語を用いて開発する機会が増えており、このようなプログラムの規模は、著しく増大する傾向にある。従って、バグのまったく存在しないプログラムを短期間で作成することは困難になっており、運用の段階でバグが顕在化して損害を発生する可能性が増大している。
【０００３】
運用段階での顕在化の多いバグとして、バッファオーバーランによるセキュリティーホールバグが挙げられる。
【０００４】
攻撃者は、プログラム実行時にスタック領域に確保された自動変数領域に、不正なデータを読み込ませ、システムダウンや、システムクラックを起こすことがある。
【０００５】
以下、図２２を用いて、バッファオーバーラン攻撃を説明する。
【０００６】
図２２（ａ）に示すソースコードでは、関数ｒｅａｄ＿ｉｎｔが内部において関数ｒｅａｄ＿ｓｔｒを呼出す。関数ｒｅａｄ＿ｓｔｒは、外部受信データを解析して、配列ｂｕｆに文字列を返す。関数ｒｅａｄ＿ｉｎｔは、関数ａｔｏｉを呼出し、得られた文字列を数値に変換し、変換後の数値を戻り値とする。
【０００７】
より詳しくは、関数ｒｅａｄ＿ｉｎｔは、図２２（ｂ）に示すようにスタック上に確保された配列ｂｕｆ内に文字列を受け取り、ＡＮＳＩ標準関数ａｔｏｉが配列ｂｕｆ内の文字列を数値に変換し、その数値がリターンパラメータ用のレジスタに格納され、スタック上に存在する呼び出し元のアドレスの参照により、利短処理が実行される。
【０００８】
ここで、関数ｒｅａｄ＿ｓｔｒの仕様として、３２バイト以上の文字列を返さないものとする。
【０００９】
ところが、関数ｒｅａｄ＿ｓｔｒ内部のロジックが複雑であり、特殊なデータが送付されると、３２バイトを超える任意のデータが、関数ｒｅａｄ＿ｉｎｔの配列ｂｕｆに返されてしまうバグが潜在していたと仮定する。
【００１０】
このような場合、攻撃者は、図２２（ｃ）に示すように、攻撃コードのアドレスが関数ｒｅａｄ＿ｉｎｔのスタックに返されるようなデータを作成して送りつけることができる。
【００１１】
この場合、関数ｒｅａｄ＿ｉｎｔが関数ｒｅａｄ＿ｓｔｒを呼出した後、スタックに攻撃コードのアドレスが上書きされる。そして、関数ａｔｏｉがダミー部分を数値に変換した後、関数ｒｅａｄ＿ｉｎｔは呼び出し元関数では無く攻撃コードにリターンしてしまう。
【００１２】
こうなると、攻撃者は、攻撃コードにより任意のコードを実行可能となる。また、任意のコード実行ができなくとも、リターンアドレスが不正になるとプログラムは暴走し、シ

ステムダウンが誘発されるおそれが大きい。

【００１３】

現時点において主たる攻撃対象となっているのは、ネットワークに接続されたサーバー装置やパーソナルコンピュータ装置であり、システムのクラッキング、ウイルスの感染経路、ＤｏＳ（ｄｅｎｉａｌ　ｏｆ　ｓｅｒｖｉｃｅ　ａｔｔａｃｋ）の踏み台などの問題が頻発している。

【００１４】

また近い将来、ネットワークに接続される組み込み機器（例えば、携帯電話やネットワーク対応家電品など）は、ますます増加するものと予想される。

【００１５】

組み込み機器では、ハードウェア制御や高速化のためＣ言語利用率が極めて高く、且つ、Ｃ言語を利用すると障害時においてプログラムを修正することは困難である。

【００１６】

また、ネットワークに接続するために同一のソフトウェアモジュールが多用されるため、一度セキュリティーホールが発覚した場合、攻撃者は、例えばルートＤＮＳサーバーに正規リクエストを繰り返す自己増殖プログラムを散布するなどの手法で、容易に世界規模でのネットワーク障害を誘発させることができる。

【００１７】

ネットワークを副次的に利用する、水道、ガス、電気などのライフライン、医療、軍事など、社会性を帯びたシステムが、このような攻撃を受ける可能性がある。

【００１８】

例えばインターネット接続可能な携帯電話や、ＴＶなどの装置にメール機能やＷｅｂブラウザ機能等を搭載する場合、受信パケットは、通常何階層ものプロトコルスタックモジュールで処理され、さらにアプリケーションで処理され、表示用の複数段の表示制御ソフトウェアの処理などを経て表示される。

【００１９】

このように、何度もデータ解析処理が行われるが、このうち１つでも異常データに対する処理に不完全なものがあれば、セキュリティーホールとなり得る。

【００２０】

正常なデータと異なり、異常なデータのパターンは、ほぼ無限に組み合わせ可能であり、加えてタイミングやトラフィックの混雑によって、ネットワークの挙動が異なる場合があるため、完全な検証は極めて困難である。

【００２１】

さらに開発規模が増大すると、人為的なバグ、すなわち悪意を持ったプログラマが開発に参入し、意図的にセキュリティーホールを作成することも考慮に入れなければならない。

【００２２】

実際、セキュリティーの必要なソフトウェア開発の下請け業者に、テロ組織の要員が含まれていた事例も存在する。

【００２３】

このような場合、セキュリティーホールは、単にバッファオーバーフローのチェック機能の漏れや、チェック値の誤りを混入させるだけで良いから、レビューや検証による抽出で発見できない可能性も高く、発見できたとしても悪意によるものかどうか判定困難である。

【００２４】

従って、プログラムソースコードを記述する際にプログラマが意識しなくとも、バグが顕在化したときに復帰できる機構をシステムに備える必要がある。

【００２５】

（従来の技術１）

プロセッサ上で実行されるプログラム障害発生時に復帰する手法として、プログラムの

正常動作中、定期的にチェックポイントを通過させ、状態を保存しておき、プログラム障害が検出された場合、復帰処理実施後、最後に通過したチェックポイントにロールバックして処理を再開する方法がある（例えば特許文献１参照）。

【００２６】

このエラー復帰方法は、Ｃ言語で書かれたプログラムに対しても適用可能である。この方法を、ある程度以上の精度を持つエラー検出手段と併せて適用すれば、プログラムバグが運用段階で顕在化した場合でも、システムをダウンさせずに処理続行ができるという利点がある。

【００２７】

（従来の技術２）

また、Ｃ言語におけるメモリアクセス違反検出手法として、コンパイルの段階でポインタにアクセス可能な範囲情報を保持させ、アクセス毎にサブルーチンを呼び出して範囲チェックを行うようにソースコードを加工することで、既存Ｃ言語プログラムに、アクセス違反検出機構を埋め込む方法がある（例えば特許文献２参照）。

【特許文献１】特表平９－５０９５１５号公報

【特許文献２】特開平７－２２５７０３号公報

【発明の開示】

【発明が解決しようとする課題】

【００２８】

（課題１）

従来の技術１の構成では、プログラムの構成を、定期的にチェックポイントを通過するように設定しなければならないため、プログラムの構造によっては、採用が困難な場合がある。

【００２９】

（課題２）

また、従来の技術１で述べる構成では、最後に通過したチェックポイントから障害発生までの間の処理が２回実施されてしまう。

【００３０】

従って、データを扱うだけのプログラムであれば、復帰処理過程である程度整合性を取ることができるが、通信制御などプログラムでは、同一のパケットを２度送出してしまったり、障害時に受信中だったパケットを損失してしまうなど副作用が発生するおそれがある。

【００３１】

そこで本発明は、障害復帰機能を強化した情報処理装置を提供することを目的とする。

【課題を解決するための手段】

【００３２】

請求項１記載の情報処理方法は、メモリのアクセス違反を検出しながら通常の処理を行うステップと、アクセス違反が検出されたら、アドレスに関連付けられた範囲情報と障害復帰情報とに基づいて復帰が可能か否かを判定するステップと、復帰が可能なら障害復帰を行うステップとを含む。

【００３３】

この構成において、範囲情報と障害復帰情報とを、アドレスに関連付けることにより、障害復帰情報を用いて、障害個所のメモリ領域のデータタイプを特定できる。

【００３４】

請求項２記載の情報処理方法では、復帰が可能でないなら停止処理を行うステップをさらに含む。

【００３５】

この構成により、障害とは無関係なデータの破壊や、システムダウン等の危険を回避できる。

【００３６】

請求項３記載の情報処理方法では、障害復帰情報は、固定サイズ属性と可変サイズ属性とを有する。

【００３７】

この構成により、一定サイズ内に入っていなければならない変数あるいはその変数を指すポインタと、サイズの変更が許される変数あるいはその変数を指すポインタとを、異なる態様で取り扱うことができる。

【００３８】

請求項４記載の情報処理方法では、可変サイズ属性は、上方伸長属性と下方伸長属性とを有する。

【００３９】

この構成により、サイズの変更が許される方向（アドレスにおける方向）が異なる変数あるいはその変数を指すポインタを、方向により異なる態様で取り扱うことができる。

【００４０】

請求項５記載の情報処理方法では、停止処理を行うステップにおいて、保存可能なデータがあれば、このデータを保存する。

【００４１】

この構成により、障害発生後、処理を再開あるいは検討する際に、保存したデータを利用できる。

【００４２】

請求項６記載の情報処理方法では、障害復帰を行うステップにおいて、読み／書きのアクセス種別を判定し、判定結果に基づいて異なる内容の障害復帰を行う。

【００４３】

この構成により、アクセス種別に応じて、異なる態様で障害復帰できる。

【００４４】

請求項７記載の情報処理方法では、アクセス種別が「書き」であるとき、そのまま障害復帰し、アクセス種別が「読み」であるとき読み出し先に所定値を格納して障害復帰する。

【００４５】

この構成により、アクセス種別が「書き」であるときは危険なメモリアクセスを回避し、「読み」であるときは読み出し側の不具合（例えば、暴走等）を回避できる。

【００４６】

請求項８記載の情報処理方法では、判定するステップにおいて、上方伸長属性があり、かつ、アクセス違反が下方である場合、復帰は可能でないと判定する。

【００４７】

請求項９記載の情報処理方法では、判定するステップにおいて、下方伸長属性があり、かつ、アクセス違反が上方である場合、復帰は可能でないと判定する。

【００４８】

請求項１０記載の情報処理方法では、判定するステップにおいて、固定サイズ属性がある場合、復帰は可能でないと判定する。

【００４９】

これらの構成により、サイズ変更が不適な場合、復帰を取り止めて危険を回避できる。

【００５０】

請求項１１記載の情報処理方法では、障害復帰を行うステップにおいて、アクセス違反した領域とは別の領域を確保し、確保した別の領域にアクセスする。

【００５１】

この構成により、別の領域へアクセスし、危険を回避できる。

【００５２】

請求項１２記載の情報処理方法では、障害復帰情報は、ターミネータ属性を有し、ターミネータ属性がデータの終端にターミネート値を有することを示す場合、障害復帰を行うステップにおいて、データの末尾にターミネート値を付加して障害復帰する。

【００５３】
この構成により、ターミネート値に依存するプロセス等の安全性を向上できる。
【発明の効果】
【００５４】
本発明によれば、情報処理装置の障害復帰機能を強化でき、動作安定性を向上できる。
【発明を実施するための最良の形態】
【００５５】
以下図面を参照しながら、本発明の実施の形態を説明する。まず具体的な構成の説明に先立ち、本発明らによる考察を記述する。
【００５６】
さて、従来の技術１において、チェックポイントまで戻らずに、障害復旧処理後に直接障害発生個所に戻ることが可能となれば、チェックポイントの設定が不要となり課題１が解決し、また、同じ部分を２度実行しないため課題２も解決する。
【００５７】
そこで、Ｃ言語で記述されたプログラムの運用段階における障害において、障害復旧処理後に直接障害発生個所に戻るシステムが考えられる。
【００５８】
障害復旧処理後に直接障害発生個所に戻る為には、まず、エラー検出の段階で次の条件がそろっている必要がある。
（条件１）障害個所以外のデータを破壊する前に障害を検出できること
（条件２）障害検出後、障害個所のメモリ領域が一意に特定できること
（条件３）障害検出後、障害個所のメモリ領域の、Ｃ言語の意味でのデータタイプが特定できること
（条件４）障害検出後、障害を検出した個所、またはその直後から処理再開できること
（条件１）、（条件２）、（条件４）を満たすエラー検出手段として従来の技術２の方法がある。
【００５９】
この方法では、専用のコンパイラを使用することが前提となる。
【００６０】
しかしながら、組み込み機器では、出荷前にＲＯＭに書き込まれたプログラム以外を実行することは無く、汎用的なアプリケーションソフトウェアの実行を想定したサーバー機やパーソナルコンピュータとは前提条件が異なる。
【００６１】
組み込みシステムでは、製造者が利用するコンパイラ以外のコンパイラでコンパイルされたコードがＲＯＭに配置されることは無い。
【００６２】
従って、完成したＣ言語ソースコードをＲＯＭに焼きこむためにコンパイルする段階で、障害復帰機能を組み込む事に問題は無い。
【００６３】
しかしながら、従来の技術２は、主にプログラム運用前の検証工程におけるバグの早期発見を目的としており、そのまま運用時に利用することも可能であるが、以下の課題が問題になる場合もある。
（課題ａ）チェック処理がソフトウェア実行であるため処理量が著しく増大する
（課題ｂ）チェック区間で割り込みが発生し、ポインタが変更されると整合性が取れなくなるため、割り込み禁止でチェックせねばならず、最大割り込み禁止時間が増大し、組み込みシステムではシステム破綻を引き起こす場合がある。
【００６４】
しかしながら、（課題ａ）、（課題ｂ）は、次のようにすれば解決可能である。
【００６５】
まず、従来の技術２と同様に、コンパイラやライブラリに機能追加を施してポインタに

対して、ポインタの指すデータに対してアドレスだけではなく割り当てられたメモリ範囲の情報を付加する。簡略化するには、ポインタ型そのものを型拡張して、アドレスと範囲情報の両者を保持できるデータ型とすれば良い。

【００６６】

この際、ポインタ型のサイズが増加するが、ＡＮＳＩ－Ｃ仕様においてポインタのデータ型を限定していないため、Ｃ言語の仕様を満たしたまま（Ｃ言語の仕様に違反せずに）実施可能である。

【００６７】

この構成にあわせて、図１に示すようなプロセッサを用意すると良い。

【００６８】

図１は、本発明の検討例におけるプロセッサ装置のブロック図である。

【００６９】

図１に示すように、このプロセッサ装置１０は、アドレスバス５及びデータバス６を介して、メモリ７やＩ／Ｏ装置８等に接続される。そして、プロセッサ装置１０は、次の構成要素を備える。

【００７０】

命令処理部１は、データバス６を経由して、メモリ７上のプログラムを読み込んで逐次実行したり、Ｉ／Ｏ装置８と入出力を行う。

【００７１】

ここで、本検討例では、命令処理部１と、アドレスバス５との間に、ＭＭＵ（メモリマネージメントユニット）４を設け、命令処理部１とＭＭＵ４とは、論理アドレスを用いて入出力をし、ＭＭＵ４は、論理アドレス／物理アドレスの変換を行うことにしているが、例えば、ＭＭＵ４を省略し、命令処理部１が、物理アドレスを入出力するようしても差し支えない。

【００７２】

さて、本検討例のプロセッサ装置１０には、通常のプロセッサ装置の要素に加えて、次のレジスタ（通常、汎用レジスタで構成すればよい。）が設けられている。

【００７３】

まず、アドレスレジスタ２は、図１のプロセッサでは、アドレスレジスタから拡張したものであって、アドレスレジスタ２は、通常のアドレスレジスタが有するアドレスを保持する領域に加え、このアドレスに関する範囲情報を格納する領域を備えている。

【００７４】

また、命令処理部１は、アドレスレジスタ２に対し、拡張されたポインタ型を１命令でロードストア可能な命令を処理できるものとする。

【００７５】

１命令でロードストアするため、アドレスと範囲情報とが割り込みによって分断されること無くアトミックに操作可能であり、（課題ｂ）を解決できる。

【００７６】

図１において、アクセス違反検出部３は、比較器から構成される。アクセス違反検出部３は、命令処理部１が、アドレスレジスタ２に係るポインタを利用して、メモリ７をアクセスする際、命令処理部１の出力（本例では、論理アドレス値）と、アドレスレジスタ２の範囲情報とを入力し、これらの比較演算を行い、メモリ７の範囲違反があるときに、例外信号を命令処理部１へ出力する。

【００７７】

このメモリ７のアクセスは、直接アドレッシングでも良いし、インデックス値の加減算を行う間接アドレッシングでも良い。

【００７８】

このプロセッサアーキテクチャにより、範囲情報が不可分に関連付けられたポインタを、範囲チェックをしながら、１機械語命令により、高速かつアトミックに、扱うことが可能となる。即ち、ソフトウェア処理の増大なしに、アクセス違反を検出でき、（課題ｂ）

を解決できる。

【００７９】

しかも、ＯＳなどプロセッサの特権モードで動作するプログラムの介在無しに、プログラム自身で、例外を発生させ、不適切なメモリアクセスをトラップできる。

【００８０】

また、プログラムカウンタ９には、命令処理部１によって読み書きされ、かつ、プログラムにおける実行アドレス値が格納される。命令処理部１は、プログラムカウンタ９に新たな実行アドレス値を格納する処理を、１機械語命令において実行する。

【００８１】

この処理は、プログラムカウンタ９の実行アドレス値を、インクリメントする場合や、ＪＵＭＰあるいはＣＡＬＬ命令のような、分岐命令を実行する際に、行われる。

【００８２】

次に、図２を参照しながら、このような構成におけるエラー検出の過程を説明する。

【００８３】

図２（ａ）のソースコードでは、関数ｍａｉｎが、ｉｎｔ型変数を３個分しかない配列ａへのポインタを関数ｆｏｏに渡し、関数ｆｏｏがバッファをオーバーして４バイトのデータを書き込もうとする。

【００８４】

まずコンパイラは、図２（ａ）のソースコードの３行目をコンパイルする時に、配列ａのサイズが「３」であることを認識する。次に、コンパイラは、５行目において、配列ａを指すポインタを生成して関数ｆｏｏにパラメータとして渡すコードを生成するが、コンパイラは、配列ａのアドレスだけでなく、配列ａの割り当てられるメモリ範囲を示す範囲情報（上限アドレス、下限アドレス）を含んだポインタを生成する。

【００８５】

コンパイル済みのコードが実行されると、関数ｆｏｏのｆｏｒループに対応するコードの実行時にループの４回目（ソースコードの１４行目，ｉ＝４のとき）において、図２（ｄ）に示すように、ポインタＰは配列ａの範囲を超えてアクセスしようとする。この際、ポインタ情報に範囲情報が含まれており、かつ、アドレスレジスタ２に、この範囲情報が格納されているから、アクセス違反検出部３がアクセス違反を検出し、例外信号を命令処理部１へ出力する。これにより、範囲を超えたアクセスを検出することが可能となる。

【００８６】

以上の構成により、（条件１）、（条件２）、（条件４）は満たせるので、本発明では残る（条件３）を満たすために、コンパイラを利用してポインタ型に、障害復帰情報をさらに付加する。次に、この付加の原理を説明する。

【００８７】

例として図３（ａ）及び図３（ｂ）に示すソースファイルと、図３（ｃ）に示す障害復帰情報ファイルを考える。

【００８８】

図３（ａ）のソースファイルの８行目において、関数ｍａｉｎは、図３（ｂ）のように定義される関数ｉｎｐｕｔ＿ｓｔｒを呼出して、配列ｂｕｆに文字列を取得する。そして、関数ｍａｉｎは、１０行目～１１行目において配列ｂｕｆの文字列長をカウントし、１３行目にて結果を標準出力に出力するものである。

【００８９】

コンパイルの段階で、ソースコードとは別に、図３（ｃ）に示すような障害復帰情報ファイルを用意し、障害復帰情報ファイルにおいてオブジェクト名、データ型、変数名などを指定し、ソースコード上のデータに対して障害復帰用の属性を付加する。なお、ここでのコンパイルとは、ソースから実行コードを出力する一連の過程を指しており、プリプロセス、リンクなども含んで広い意味でのコンパイルを指している。ここで、図３（ｃ）の障害復帰情報ファイルは、元のソースコードそのものとは異なる存在であり、コンパイラにより評価されるから、コンパイラ擬似命令やマクロに似た存在であるということができ

る。

【００９０】

図３（ｃ）の障害情報ファイルでは、ｍａｉｎ．ｃ（図３（ａ））内のｃｈａｒ型の配列に対して、上方伸長属性（ＵＰＰＥＲ）を付加している。本形態では他に、下方伸長属性（ＬＯＷＥＲ）と固定サイズ（ＦＩＸＥＤ）とを取扱う。

【００９１】

図４（ａ）は関数ｍａｉｎから関数ｉｎｐｕｔ＿ｓｔｒを呼び出した際のスタックメモリの内容を示し、図４（ｂ）は、引数として渡されるポインタのデータ構造を示す。

【００９２】

関数呼び出しに際し、配列ｂｕｆを指すポインタ（図４（ｂ））が生成されるが、生成されるポインタ型には配列ｂｕｆのアドレスのみならず、範囲情報（アドレス上限、アドレス下限）と障害復帰情報（ＵＰＰＥＲ）が含まれている。

【００９３】

ｃｈａｒ型の配列ｂｕｆは、関数ｍａｉｎによりスタックメモリに図４（ａ）に示すように確保される変数であるが、図４（ｂ）のポインタで範囲情報が渡されるために、関数ｉｎｐｕｔ＿ｓｔｒは、メモリアクセス範囲違反を適切に監視できる。

【００９４】

ここで、関数ｉｎｐｕｔ＿ｓｔｒにて、設計想定外のデータとして３２バイト以上のデータが入力された場合、図３（ｂ）のソースコード５行目のｗｈｉｌｅループは３２回以上実施され、図３（ｂ）のソースコード６行目のアクセスでアクセス違反が発生し、アクセス違反検出部３は、命令処理部１に例外信号を出力する。

【００９５】

命令処理部１で実行される例外処理ハンドラを用意し、アクセス違反例外のリターンアドレスを調べれば、エラー発生個所の命令が判別できる。

【００９６】

命令を解析すれば、アクセスに用いられた操作が、書き込みか読み出しのいずれであるか、及び、この命令時のプログラムアドレスを特定できる。さらにアドレスレジスタ２から、違反時にアクセスしようとしたアドレス、範囲情報、及び障害復帰情報を取得可能である。

【００９７】

次に障害復帰情報をチェックする。図４（ｂ）の例では、障害復帰情報は、上方伸長属性（ＵＰＰＥＲ）であるから、アドレス上方に伸びる可変長のデータを格納するデータ型であると判断でき、違反時にアクセスしようとしたアドレスが範囲情報の上側かどうかをチェックすれば良い。

【００９８】

この例では、上側にアクセス違反を起こしているため、バッファオーバーランが発生していると判断できる。

【００９９】

また、この例では、アクセスが「書き込み」であるから、バッファオーバーラン用の障害復帰処理として、アクセスエラーの発生した次のアドレスにリターンし、書き込みを無効化すると良い。

【０１００】

上記操作によれば、図４（ａ）に示すスタックメモリのｂｕｆ領域以外を一切破壊することなく、関数ｉｎｐｕｔ＿ｓｔｒを終了させ得る。

【０１０１】

さて、図３（ａ）のソースコード１０行目において、関数ｍａｉｎは、ｗｈｉｌｅループを実行し、ＮＵＬ文字（ターミネート値の例）である「０」の検索を行う。

【０１０２】

関数ｉｎｐｕｔ＿ｓｔｒで読み出したデータの前半３２バイトにＮＵＬ文字が１つも含まれていない場合、このｗｈｉｌｅループは３２回以上実行され、配列ｂｕｆの領域を越

えてアクセスが発生することになり、アクセス違反が発生する。

【０１０３】

この場合、書き込み時と同様に例外処理を実行すれば良い。この際、書き込み時と異なるのは、読み出しアクセスである点だけである。

【０１０４】

読み出しアクセスの場合、障害復帰処理は、次のようにすれば良い。即ち、例外を発生した読み出しアクセスの結果に代えて、読み出し結果にターミネート値「０」を格納し、読み出しアクセス命令の次の命令にリターンする。

【０１０５】

こうすれば、関数ｍａｉｎのｗｈｉｌｅループの終了条件が満たされるため、図４（ａ）に示すスタックメモリのｂｕｆ領域以外を一切破壊せずに、ｗｈｉｌｅループを終了させ得る。

【０１０６】

本考察によれば、従来技術では、システムがクラッキングしてしまう、もしくはエラー検出しても強制終了しかできなかった攻撃又はバグに対して、プログラムの障害復帰を行うことができる。

【０１０７】

さらに、ソースコード自身にバッファオーバーフローを許すバグが意図的に埋め込まれている場合であっても、プログラムの障害復帰を行える。

【０１０８】

また、ここまでの例は、ポインタに障害復帰情報をもたせている。即ち、図５（ａ）のソースコードの７行目でパラメータ渡しが実施されるとき、図５（ｂ）に示すようにポインタ値が障害復帰情報を保持するようにしている。

【０１０９】

これにかえて、図６に示すように、図５（ａ）のソースコードの１行目で宣言される配列ａ自身に、その範囲に隣接して障害復帰情報を持たせるようにしてもよい。

【０１１０】

このようにしても、アクセス違反が発生した場合、範囲情報が取得できるから、範囲情報に隣接するアドレスを読み出せば、図５（ｂ）に示す場合と同様に、障害復帰情報を取得可能である。

【０１１１】

図５（ｂ）によると、ポインタは、しばしばレジスタ経由でパラメータ渡しに利用されるから、パラメータ受け渡しの処理負荷が増える。

【０１１２】

一方、図６によると、ポインタアクセスする可能性のある全ての変数について障害復帰情報をつける必要があるから、メモリ使用量が増えるが、ポインタ渡しの処理量が削減できるメリットがある。以上で、本考察を終える。

【０１１３】

以上の考察をふまえ、以下本発明の骨子を説明する。

【０１１４】

（発明の骨子１）

本骨子は、コンパイル装置に関する。以上の考察を具体化すると、コンパイル装置は、図８に示すようになる。

【０１１５】

即ち、コンパイル装置３０は、ポインタを取扱えるプログラム言語（例えば、Ｃ／Ｃ＋＋言語やＰａｓｃａｌ等）で記述されたソースコード２１（例えば、図３（ａ）、（ｂ）参照）を読み込み、これをコンパイルして実行コードを出力する。なお、ここでいう「コンパイル」の意味は、上記考察で述べたとおりである。

【０１１６】

また、コンパイル装置３０は、ソースコード２１をコンパイルする際に、障害復帰情報

ファイル２２（例えば、図３（ｃ））を参照する。なお、障害復帰情報は、コンパイル装置３０が認識できる形態となっていれば十分であり、必ずしもファイルの形式で提供されなくとも良い。

【０１１７】

次が、重要な点である。即ち、コンパイル装置３０は、ソースコード２１を単にコンパイルするだけでなく、次の（関連付け１）または（関連付け２）のいずれかを実施する。

（関連付け１）：ポインタについて、ポインタのアドレス、範囲情報及び障害復帰情報を不可分に関連付ける（図５（ｂ）参照）。

（関連付け２）：ポインタについてポインタのアドレス及び範囲情報を不可分に関連付け、かつ、このポインタで操作される変数に障害復帰情報を不可分に関連付ける（図６参照）。

【０１１８】

（発明の骨子２）

本骨子は、情報処理装置に関する。さらに詳しくは、この情報処理装置は、図７に示すプロセッサを備え、このプロセッサは、発明の骨子１におけるコンパイル装置３０が生成した実行コードを実行する。

【０１１９】

図７は、本発明の骨子２におけるプロセッサのブロック図である。

【０１２０】

プロセッサ２０のアドレスレジスタ１２は、アドレス情報、範囲情報、及び、障害復帰情報を格納可能である。

【０１２１】

命令処理部１は、メモリ７から読み出したプログラムコードを解釈し、実行するが、この際にメモリ７上のアドレス情報、範囲情報、及び、障害復帰情報がセットになったポインタ型をアドレスレジスタ１２にロード／ストアする命令を実行可能である。

【０１２２】

また、このプロセッサ２０は、アクセス違反検出部３を備えており、命令処理部１がアドレスレジスタ１２のアドレス情報を利用したアドレッシング命令を利用した場合、アクセス違反検出部３は、命令処理部１が出力する論理アドレスとアドレスレジスタの範囲情報を比較し、アクセスする論理アドレスが範囲外であった場合、命令処理部１に例外信号を出力する。

【０１２３】

命令処理部１は、例外信号を受け取ると、例外処理用の実行番地にその実行を分岐する。

【０１２４】

このプロセッサ２０により、割り込み処理などによってポインタのもつアドレス情報、範囲情報、及び、障害復帰情報の操作を分断され、不整合を起こすことなく、且つ、高速にエラー検出を実施することが可能となり、さらに例外処理時にアドレスレジスタ１２から障害復帰情報を取得することが可能となる。

【０１２５】

このときの本情報処理装置を機能ブロック図で示すと、図９のようになる。

【０１２６】

さて、通常処理部２３は、メモリ７に格納されるプログラムを読み込み、アドレスレジスタ１２、プログラムカウンタ９を更新しながら通常の処理を行う。

【０１２７】

障害復帰判定部２４は、通常処理部２３がアクセス違反検出部３から例外信号を入力すると呼び出され、アドレスレジスタ１２内の障害復帰情報を参照し、障害復帰の可否を判定する。

【０１２８】

復帰処理部２５は、障害復帰判定部２４が障害復帰可とする際、障害復帰判定部２４に

呼出され、障害の状況に合わせて障害復帰処理を行い、処理を通常処理部２３へ戻す。このとき、通常処理部２３は通常処理を継続し、情報処理装置は停止しない。

【０１２９】

停止処理部２６は、障害復帰判定部２４が障害復帰不可とする際、障害復帰判定部２４に呼出され停止処理を行い、処理を通常処理部２３へ戻す。このとき、通常処理部２３は通常処理を継続せず、情報処理装置は停止する。

【０１３０】

ここで、図９における通常処理部２３、障害復帰判定部２４、復帰処理部２５及び停止処理部２６は、図７における命令処理部１がメモリ７内のプログラムを実行することにより実現される。

【０１３１】

（実施の形態１）

図１０は、本発明の実施の形態１における情報処理装置のフローチャートである。

【０１３２】

以下、図９及び図１０を参照しながら、通常処理部２３がアクセス違反検出部３から例外信号を入力した後の処理を説明する。

【０１３３】

まずステップ１において、障害復帰判定部２４は、通常処理部２３から例外発生時のプログラムカウンタ９の値を取得し、アクセス違反が発生したアドレスの命令を取得する。

【０１３４】

取得した命令は、メモリアクセスを実施する命令のはずであり、読み出しか、書き込みか、どのアドレスレジスタを利用したかの情報を一意に特定できる。

【０１３５】

アドレスレジスタ１２には、ポインタ情報が格納されているため、障害復帰判定部２４は、本来アクセスすべきデータの範囲情報と障害復帰情報とを取得できる。

【０１３６】

次にステップ２において、障害復帰判定部２４は、取得した障害復帰情報をチェックする。ここでは障害復帰情報として、上方伸長属性（ＵＰＰＥＲ）と、下方伸長属性（ＬＯＷＥＲ）と、固定サイズ属性（ＦＩＸＥＤ）を例示するが、それ以外の属性を利用し、データ属性にあった復帰手段を用意することも可能である。

【０１３７】

ステップ２において、上方伸長属性（ＵＰＰＥＲ）と判断された場合、障害復帰判定部２４は、ステップ３において、アクセス違反がデータ範囲の上方に対して行われたかどうかをチェックする。アドレス上方でなかった場合は、障害復帰判定部２４は障害復帰不可と判定し、停止処理部２６を呼出す。ステップ６にて、停止処理部２６は保存可能なデータがあればメモリ７に保存し（ステップ６）、通常処理部２３システムを停止させる。

【０１３８】

アドレス上方であれば、障害復帰判定部２４は、バッファオーバーフロー（障害復帰可）と判定し、復帰処理部２５を呼出して復帰を試みる。即ち、復帰処理部２５は、ステップ８にてアクセス種別をチェックし書き込みであれば無効化し、読み出しであればステップ９にて固定値としてターミネート値（例えば「０」）を読み出す。その後通常処理部２３は、障害箇所の次のステップへ復帰する（ステップ１０）。

【０１３９】

ステップ２において、下方伸長属性（ＬＯＷＥＲ）と判断された場合、ステップ４において、障害復帰判定部２４は、アクセス違反がデータ範囲の下方に対して行われたかどうかをチェックする。アドレス下方でなかった場合は障害復帰判定部２４は障害復帰不可と判定し、復帰処理部２５を呼出す。ステップ６にて、停止処理部２６は保存可能なデータがあればメモリ７に保存し（ステップ６）、通常処理部２３は、システムを停止させる（ステップ７）。

【０１４０】

アドレス下方であれば、障害復帰判定部２４は、バッファオーバーフロー（障害復帰可）と判定し、復帰処理部２５を呼出し復帰を試みる。即ち、復帰処理部２５は、ステップ８にてアクセス種別をチェックし、書き込みであれば無効化し、読み出しであればステップ９にて固定値としてターミネート値（例えば「０」）を読み出す。その後通常処理部２３は、障害箇所の次のステップへ復帰する（ステップ１０）。

【０１４１】

ステップ２において、上方伸長属性でも下方伸長属性でも無い（ＦＩＸＥＤ）と判断された場合、障害復帰判定部２４は、障害復帰不可と判定し、復帰処理部２５を呼出す。ステップ６にて、停止処理部２６は保存可能なデータがあれば保存し（ステップ６）、通常処理部２３は、システムを停止させる（ステップ７）。

【０１４２】

（実施の形態１の効果）

以上のように、アクセス違反例外を処理することで、データの範囲を超えた書き込み処理において、無関係のデータを破壊することなく処理継続を可能にし、さらに、データの範囲を超えた読み出し処理においても、無関係のデータを読み出すことなく、安全にデータを読み出すことができる。

【０１４３】

図１１は、本発明の実施の形態１におけるコンパイル装置の機能ブロック図ある。

【０１４４】

ソースコード２１は、言語解析部３４に入力され、言語解析部３４はソースコード２１を実行コードへ翻訳する。ここで、言語解析部３４は、ポインタ生成において、変数へのアドレスと、範囲情報と、障害修復情報が入るべきコードを確保するが、格納する値は未定値のままである。

【０１４５】

また、言語解析部３４は、変数宣言部分を見つけると、変数に割り当てた領域のアドレスと範囲情報とを変数領域記憶部３５に書き込んで保存する。

【０１４６】

翻訳された実行コードは、コード生成部３６に送られる。

【０１４７】

障害復帰情報読み込み部３１は、ソースコード２１とは異なる障害復帰情報ファイル２２から情報を読み出し、障害復帰情報記憶部３２に格納する。

【０１４８】

コード生成部３６は、言語解析部３４から出力される実行コードの中の、未定位置となっているポインタの内部の値について、変数アドレスと、範囲情報について変数領域記憶部３５から読み出して格納する。

【０１４９】

また、コード生成部３６は、ポインタの指す変数の情報を検索部３３に送る。検索部３３では、入力された変数情報から、障害復帰情報記憶部３２内を検索し、変数に対応する障害復帰情報をコード生成部３６に出力する。コード生成部３６は、検索部３３から得られた検索結果としての障害復帰情報をコード中のポインタ内に書き込む。

【０１５０】

これにより、コード生成部３６は、変数アドレスと、範囲情報、障害復帰情報がポインタ部分に格納された実行コード４０を出力する。

【０１５１】

本コンパイル装置により、ソースコード２１から障害復帰情報の埋め込まれた実行コードが生成されるため、生成された実行コードを情報機器にインストールすれば、この情報機器に障害復帰機能を持たせることができる。

【０１５２】

図１２は、本発明の実施の形態１におけるコンパイル装置のフローチャートである。

【０１５３】

ステップ２０において、言語解析部３４は、ソースコード２１の先頭から順次ソース解析を実施する。

【０１５４】

ステップ２０において、変数宣言であった場合、ステップ２１にて、言語解析部３４は、変数用の領域を確保し、その情報を変数領域記憶部３５に保存する。

【０１５５】

ステップ２０において、変数からポインタを生成する命令であった場合、ステップ２２にて、言語解析部３４は、ポインタ値として、変数のアドレス、変数の範囲情報、障害復帰情報を含んだ値を生成するコードを生成する。

【０１５６】

ここでは、ポインタ内の値は決定されないため、言語解析部３４は、値が未定のコードを生成する。

【０１５７】

ステップ２０において、ポインタ演算であった場合、ステップ２３にて、言語解析部３４は、ポインタ内のアドレス値のみを演算し、範囲情報及び障害復帰情報はそのままコピーするコードを生成する。

【０１５８】

ステップ２０において、上記のいずれでもなかった場合、ステップ２４にて、言語解析部３４は、通常のコンパイラと同様に言語に対応するコードを生成する。

【０１５９】

言語解析部３４がここまでの手順をソースコード終端まで繰り返した後（ステップ２５）、コード生成部３６は、コード内の未定となっているポインタ情報のうち、アドレス情報と範囲情報を、変数領域記憶部３５に保存されている変数用の領域の情報から決定し、格納する（ステップ２６～２８）。

【０１６０】

最終工程として、コード生成部３６は、ポインタ情報内の障害復帰情報が未定となっているポインタを検索する。検索部３３は、障害復帰情報が未定となっているポインタが指し示している変数の属性を、障害復帰情報記憶部３２から検索し、一致する設定があれば、コード生成部３６は、その値を格納し、検出されなければ固定の値、例えば固定サイズを示す属性（ＦＩＸＥＤ）を設定する。この操作は、未定値がなくなるまで繰り返される（ステップ２６～３０）。

【０１６１】

なお、ここでは、ソースコードから実行可能コードを生成するまでの一連の工程を「コンパイル」と呼んでおり、狭い意味でのコンパイルやリンクなどの工程に分割しても構わない。

【０１６２】

（実施の形態１の変形例）

図１３は、図１０のステップ８以降に（図９における復帰処理部２５の処理内容）に変更を加えたものである。

【０１６３】

説明の重複を避けるため、ステップ８以降の処理のみ説明する。ステップ８にて、復帰処理部２５は、アクセス種別をチェックする。書き込みであれば、復帰処理部２５はアクセスすべきデータとは別の領域をメモリに確保し、書き込もうとしていたデータを確保した領域に書き込む（ステップ３４～３６）。

【０１６４】

読み出しであれば、ステップ３１にて、復帰処理部２５は、既に書き込もうとしていたデータをステップ３５で確保した領域が無いかチェックし、存在すればステップ３２にて復帰処理部２５は、この領域の値を読み出し、存在しなければステップ３３にて固定値としてターミネート値（例えば「０」）を読み出す。

【０１６５】

（変形例の効果）
上記のように、アクセス違反例外を処理することで、データの範囲を超えた書き込み処理において、自動的にデータサイズを拡張することが可能となる。
【０１６６】
入力サイズが見積もり値を超えた際、単に破棄するだけでは破綻してしまうようなデータに対する復帰が可能となる。
【０１６７】
（実施の形態２）
図１４は、本発明の実施の形態２におけるコンパイル装置の機能ブロック図である。図１４において、実施の形態１に係る図１１と同様の構成要素には、同一符号を付すことにより、説明を省略する。
【０１６８】
さて、実施の形態１では、図５（ｂ）に示すように、ポインタ値に障害復帰情報を持たせた（関連付け１）が、実施の形態２では図６に示すようにポインタで操作される変数に障害復帰情報を持たせる（関連付け２）。
【０１６９】
このため、実施の形態１におけるコード生成部３６（図１１参照）を、図１４に示すように、第１コード生成部３７と第２コード生成部３８とに分けている。また、言語解析部３４’の処理内容が変更されている。
【０１７０】
また、言語解析部３４’はソースコード２１において、変数宣言部分を見つけると、変数に割り当てた領域のアドレスと範囲情報とを変数領域記憶部３５に書き込んで保存すると共に、変数領域に隣接する場所に障害復帰情報が入る容量を余分に確保する。これにより、図６に示す変数領域が確保される。
【０１７１】
また、第１コード生成部３７は、言語解析部３４’から出力される実行コードの中の、未定位置となっているポインタの内部の値について、変数アドレスと、範囲情報について変数領域記憶部３５から読み出して格納する。
【０１７２】
そして、第２コード生成部３８は、第１コード生成部３７から出力された、変数アドレスと、範囲情報がポインタ部分に格納された実行コードを受け取り、変数情報を順次検索部３３に送信し、検索部３３から得られた検索結果としての障害復帰情報を変数情報に隣接する領域に格納する。
【０１７３】
他の点は、実施の形態１と同様である。
【０１７４】
図１５は、本発明の実施の形態２におけるコンパイル装置のフローチャートである。
【０１７５】
ステップ４０にて、言語解析部３４’は、ソースコード２１の先頭から順次ソース解析を実施する。
【０１７６】
ステップ４０において、変数宣言であった場合、言語解析部３４’は、変数用の領域と変数用の領域に隣接する障害復帰情報格納用の領域を確保し、その情報を保存する（ステップ４１、４２）。
【０１７７】
ステップ４０において、変数からポインタを生成する命令であった場合、ステップ４３にて、言語解析部３４’は、ポインタ値として、変数のアドレス、変数の範囲情報、障害復帰情報を含んだ値を生成するコードを生成する。
【０１７８】
ここでは、ポインタ内の値は決定されないため、言語解析部３４’は、値が未定のコー

ドを生成する。

【０１７９】

ステップ４０において、ポインタ演算であった場合、ステップ４４にて、言語解析部３４’は、ポインタ内のアドレス値のみを演算し、範囲情報及び障害復帰情報はそのままコピーするコードを生成する。

【０１８０】

ステップ４０において、上記のいずれでもなかった場合、ステップ４５にて、言語解析部３４’は従来のコンパイラと同様に言語に対応するコードを生成する。

【０１８１】

言語解析部３４’がここまでの手順をソースコード終端まで繰り返した後（ステップ４６）、ステップ４７にて、第１コード生成部３７は、コード内の未定となっているポインタ情報のうち、アドレス情報と範囲情報を、保存されている変数用の領域の情報から決定し、格納する。

【０１８２】

最終工程として、第２コード生成部３８は、データ情報に付随する情報内の障害復帰情報が未定となっているポインタを検索する。検索部３３は、障害復帰情報が未定となっているポインタが指し示している変数の属性を、障害復帰情報記憶部３２から検索し、一致する設定があれば、第２コード生成部３８は、その値を格納し、検出されなければ固定の値、例えば固定サイズを示す属性（ＦＩＸＥＤ）を設定する。この操作は、未定値がなくなるまで繰り返される（ステップ４８～５１）。

【０１８３】

なお、ここでは、ソースコードから実行可能コードを生成するまでの一連の工程を「コンパイル」と呼んでおり、狭い意味でのコンパイルやリンクなどの工程に分割しても構わない。

【０１８４】

（実施の形態３）

図１６は、本発明の実施の形態３における、情報処理装置のフローチャートである。なお、実施の形態３の機能ブロック図は、実施の形態１と同じく図９である。以下、実施の形態１との相違点を中心に説明する。

【０１８５】

ステップ２において、障害復帰判定部２４は、取得した障害復帰情報をチェックする。ここでは障害復帰情報として、上方伸長属性（ＵＰＰＥＲ）と、下方伸長属性（ＬＯＷＥＲ）と、固定サイズ属性（ＦＩＸＥＤ）を例示するが、それ以外の属性を利用し、データ属性にあった復帰手段を用意することも可能である。

【０１８６】

ステップ２において、上方伸長属性（ＵＰＰＥＲ）と判断された場合、実施の形態１と同様に、ステップ５２にて、障害復帰判定部２４は、上方伸長属性の復帰処理を実施する。更に、ステップ５４にて、障害復帰判定部２４は、データにターミネータ属性が付されている場合、ステップ５５にてデータ内にターミネータデータが存在するかどうか検索し、ステップ５６にてデータが存在しなければ、ステップ５７にて、障害復帰判定部２４は、データ末尾にターミネータデータ（例えば、「０」）を追加する。これにより、失なわれてしまったターミネータデータを復元でき、データの信頼性を向上できる。

【０１８７】

なお、ステップ５３において、下方伸長属性（ＬＯＷＥＲ）と判断された場合、障害復帰判定部２４は、実施の形態１と同様の復帰処理を実施し、上方伸長属性でも下方伸長属性でも無い（ＦＩＸＥＤ）と判断された場合も、障害復帰判定部２４は、実施の形態１と同様の処理を実施する。

【０１８８】

（実施の形態３の効果）

例えばＣ言語において、文字列のようなデータは、その終端にＮＵＬ文字を付加するこ

とでデータの終端を表している。

【０１８９】

しかしながら、不正なデータなどによりデータ構造が破壊されると、データ内にＮＵＬ文字が存在しないことになり、本来、アクセスしてはならないアドレスのデータを書き換えてしまうなど、後のデータ操作で深刻な障害を引き起こしやすくなる。

【０１９０】

本形態によれば、障害発生時に文字列のようなデータ構造を正規の形式に強制的に正規化することが可能となり、副次的な障害発生を抑制できる。

【０１９１】

（実施の形態４）

図１７は、本発明の実施の形態４におけるコンパイル装置のブロック図であり、図１８は、その装置のフローチャートである。

【０１９２】

さて、実施の形態１～３では、ソースコード２１に直接変更を加えずに、コード生成を行った。

【０１９３】

一方、本形態では、コンパイル装置の内部において、ソースコード２１に情報を付加し中間コードを生成する。例えば、図３（ａ）、（ｂ）に示すソースコードがあり、図３（ｃ）に示す障害復帰情報ファイルがあるときに、図３（ａ）のソースコードに基づいて図１９（ａ）の中間コードを生成し、図３（ｂ）のソースコードに基づいて図１９（ｂ）の中間コードを生成する。なお、中間コードの生成自体は、例えば文字列の置換等、周知技術を用いて簡単に実装できる。

【０１９４】

図１９（ａ）の中間コード５行目、６行目に、「ＵＰＰＥＲ」、「ＦＩＸＥＤ」というキーワードが挿入されており、同様に、図１９（ｂ）の中間コード３行目に「ＦＩＸＥＤ」というキーワードが挿入されている。

【０１９５】

図１７において、障害復帰情報付加部３９は、ソースコード２１を入力し、ソースコード２１に障害復帰情報を付加した中間コードを生成する。その他の点は、実施の形態１と同様である。

【０１９６】

ここで、実施の形態１～３及び本形態において、プログラマ自身がソースコード２１に直接手を加えなくても、障害復帰情報を追加できる点に注目されたい。よくある状況であるが、既存のソースコードが多量存在し、これらに基づく実行コードを、本発明の障害復帰機能を持つバージョンへ一括して変更したいような場合、この点は極めて有利になる。

【０１９７】

次に、図１８を参照しながら、本形態のコンパイル装置の動作を説明する。まずステップ７０にて、障害復帰情報読み込み部３１は障害復帰情報ファイル２２を読み込み、障害復帰情報記憶部３２に障害復帰情報をセットする。

【０１９８】

次に、ステップ７１にて、障害復帰情報付加部３９は、ソースコード２１を１ステップ（例えば、１行等）読込み、ステップ７２にて、このステップに変数宣言があるかどうかチェックする。なければ、ステップ７５へ処理が移る。

【０１９９】

変数宣言があれば、ステップ７３にて、障害復帰情報付加部３９は検索部３３に変数情報を出力し、この変数情報が示す変数の障害復帰情報を検索する。この情報が存在すれば、ステップ７４にて、障害復帰情報付加部３９は、変数宣言に障害復帰情報を付加し、ステップ７５へ処理が移る。

【０２００】

ステップ７５において、障害復帰情報付加部３９はこのステップについての中間コード

をテンポラリファイルとして外部（例えば、ハードディスク装置の一定領域等）へ出力する。ステップ７６にて、障害復帰情報付加部３９は、ソースコード２１の終端に至っていないことを確認し、ソースコード２１の終端に至るまで、ステップ７１～７６の処理を繰り返す。

【０２０１】

これにより、図１９（ａ）、（ｂ）に示すような中間コードが生成されていることになる。

【０２０２】

中間コードが生成された後は、実施の形態１において、「ソースコード２１」とあるのを「中間コード」と読み替えた処理が行われる（ステップ７７～８３）。

【０２０３】

本形態によっても、実施の形態１と同様の効果がある。

【０２０４】

（実施の形態５）

図２０は、本発明の実施の形態５における障害情報監視システムのブロック図である。

【０２０５】

情報処理装置５０は、図７に示すプロセッサ２０、アドレスバス５、データバス６、メモリ７及びＩ／Ｏ装置８を備える。メモリ７には、図９に示す機能を実装したプログラム（通常のアプリケーションコードの他、障害復帰コードを含むもの）がロードされている。

【０２０６】

Ｉ／Ｏ装置８は、伝送路５５を介して情報センター６０のＩ／Ｏ装置６１と接続している。情報センター６０は、Ｉ／Ｏ装置６１の他、障害情報を記録する記憶装置６２と、障害情報を表示する表示装置６３とを有する。

【０２０７】

情報処理装置５０において、アプリケーションコードの実行時に障害が発生すると、プロセッサ２０は、メモリ７内の障害復帰コードを実行する。

【０２０８】

障害復帰コードが実行されると、障害内容の情報が、伝送路５５を介して情報センター６０に送信される。

【０２０９】

情報センター６０では、障害内容が記録装置６２に記録され、表示装置６３が障害の内容を表示する。

【０２１０】

（実施の形態５の効果）

このようなシステム構成によって、例えば携帯電話のような各ユーザーに販売される情報処理装置の運用段階での障害情報を収集することが可能となる。

【０２１１】

従って、情報処理装置の次回出荷分のロットで障害個所を修正したり、他の開発において類似の障害を残さないように開発方法を改善するのに役立てることができる。

【０２１２】

（実施の形態６）

図２１は、本発明の実施の形態６における障害情報監視システムのブロック図である。本形態は、実施の形態５に対し、情報センター６０に、変更を加えたものである。

【０２１３】

情報センター６０では、障害に対する復帰方法の情報を入力装置６５から入力し、記憶装置６２に蓄積しておくことが可能である。

【０２１４】

情報処理装置５０において、アプリケーションの実行時に障害が発生すると、プロセッサ２０は、障害復帰コードを実行する。これにより、障害内容の情報が、伝送路５５を介

して情報センター６０に送信され、情報センター６０では、検索装置６４が記憶装置６２を検索し、今回の障害に対応する復帰方法の情報を送り返す。

【０２１５】

情報処理装置５０では、送り返されてきた復帰方法の情報に従って復帰処理を行う。

【０２１６】

（実施の形態６の効果）

このようなシステム構成によって、例えば携帯電話のような各ユーザーに販売される情報処理装置の運用段階での障害復帰手段の変更が可能となる。

【０２１７】

事前に組み込んだ障害復帰機構のみで対応できないトラブルが発生した場合でも、情報センター側で対処方法を登録することで、トラブルを収拾することが可能となる。

【産業上の利用可能性】

【０２１８】

本発明に係る方法は、例えば、ポインタを扱える言語を用いたプログラム開発の分野等において広く好適に利用できる。

【図面の簡単な説明】

【０２１９】

【図１】本発明の考察におけるプロセッサ等のブロック図

【図２】（ａ）本発明の考察におけるソースコードの例示図　（ｂ）本発明の考察におけるポインタアクセスの説明図　（ｃ）本発明の考察におけるポインタアクセスの説明図　（ｄ）本発明の考察におけるポインタアクセスの説明図

【図３】（ａ）本発明の考察におけるソースコードの例示図　（ｂ）本発明の考察におけるソースコードの例示図　（ｃ）本発明の考察における障害復帰ファイルの例示図

【図４】（ａ）本発明の考察におけるスタックメモリの説明図　（ｂ）本発明の考察におけるポインタの説明図

【図５】（ａ）本発明の考察におけるソースコードの例示図　（ｂ）本発明の考察におけるポインタと変数の関係図

【図６】本発明の考察におけるポインタと変数の関係図

【図７】本発明の実施の形態１におけるプロセッサ等のブロック図

【図８】本発明の実施の形態１におけるコンパイル装置の概念図

【図９】本発明の実施の形態１における情報処理装置の機能ブロック図

【図１０】本発明の実施の形態１における情報処理装置のフローチャート

【図１１】本発明の実施の形態１におけるコンパイル装置の機能ブロック図

【図１２】本発明の実施の形態１におけるコンパイル装置のフローチャート

【図１３】本発明の実施の形態１（変形例）におけるコンパイル装置のフローチャート

【図１４】本発明の実施の形態２におけるコンパイル装置の機能ブロック図

【図１５】本発明の実施の形態２におけるコンパイル装置のフローチャート

【図１６】本発明の実施の形態３における情報処理装置のフローチャート

【図１７】本発明の実施の形態４におけるコンパイル装置の機能ブロック図

【図１８】本発明の実施の形態４におけるコンパイル装置のフローチャート

【図１９】（ａ）本発明の実施の形態４におけるソースコードの例示図　（ｂ）本発明の実施の形態４におけるソースコードの例示図

【図２０】本発明の実施の形態５における障害情報監視システムのブロック図

【図２１】本発明の実施の形態６における障害情報監視システムのブロック図

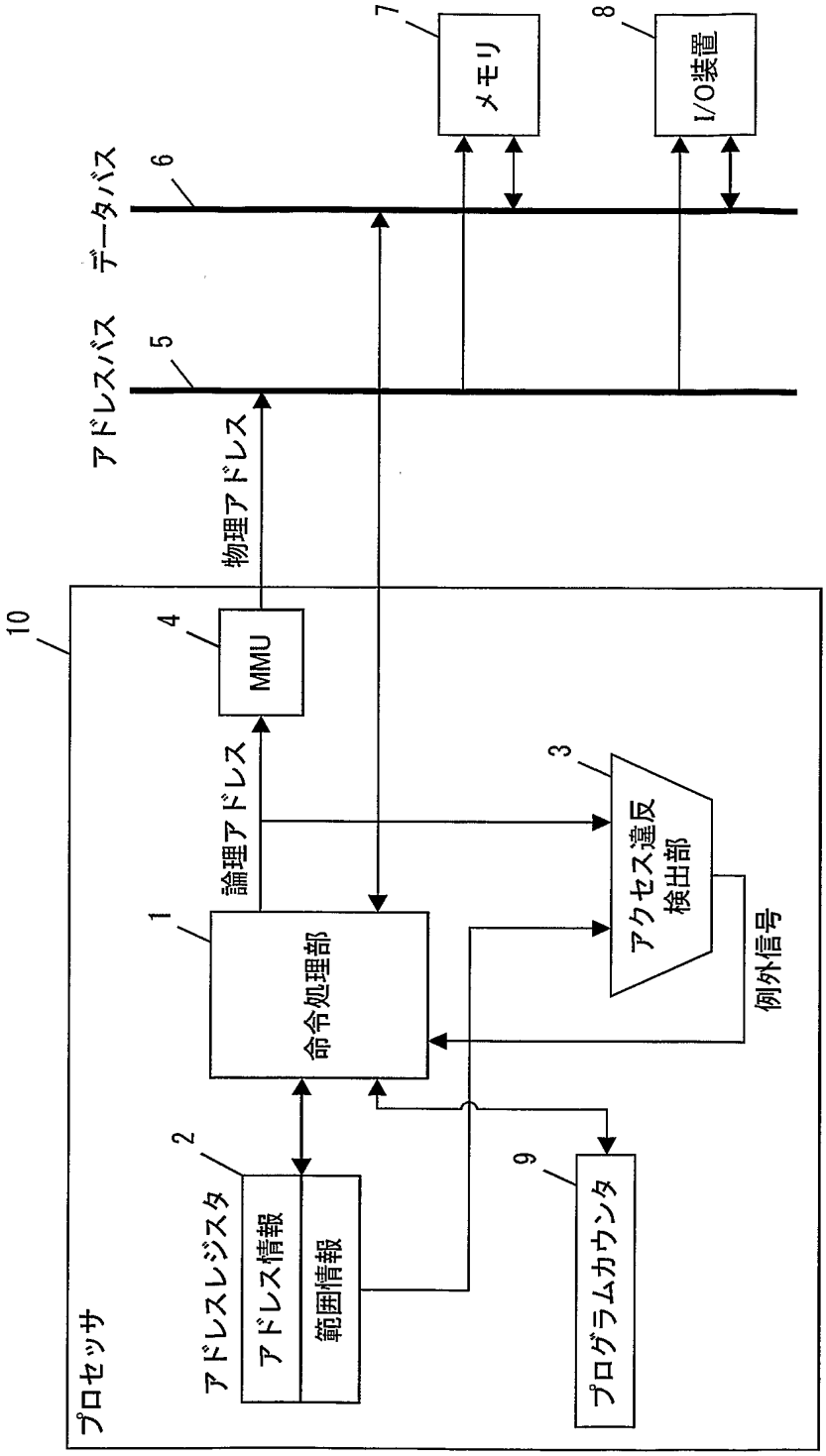
【図２２】（ａ）従来のソースコードの例示図　（ｂ）従来のスタックメモリの説明図　（ｃ）攻撃用データの例示図

【符号の説明】

【０２２０】

１　命令処理部
２　アドレスレジスタ
３　アクセス違反検出部
４　メモリマネージメントユニット
５　アドレスバス
６　データバス
７　メモリ
８　Ｉ／Ｏ装置
９　プログラムカウンタ
１０　プロセッサ
１２　アドレスレジスタ
２０　プロセッサ
２１　ソースコード
２２　障害復帰情報ファイル
２３　通常処理部
２４　障害復帰判定部
２５　復帰処理部
２６　停止処理部
３０　コンパイル装置
３１　障害復帰情報読み込み部
３２　障害復帰情報記憶部
３３　検索部
３４、３４’　言語解析部
３５　変数領域記憶部
３６　コード生成部
３７　第１コード生成部
３８　第２コード生成部
３９　障害復帰情報付加部
４０　実行コード
５０　情報処理装置
５５　伝送路
６０　情報センター
６１　Ｉ／Ｏ装置
６２　記憶装置
６３　表示装置
６４　検索装置
６５　入力装置

【書類名】図面
【図１】


7
メモリ
8
I/O装置
データバス
6
アドレスバス
5
物理アドレス
10
4
MMU
論理アドレス
3
アクセス違反
検出部
例外信号
1
命令処理部
2
アドレスレジスタ
アドレス情報
範囲情報
9
プログラムカウンタ
プロセッサ

【図２】

（ａ）

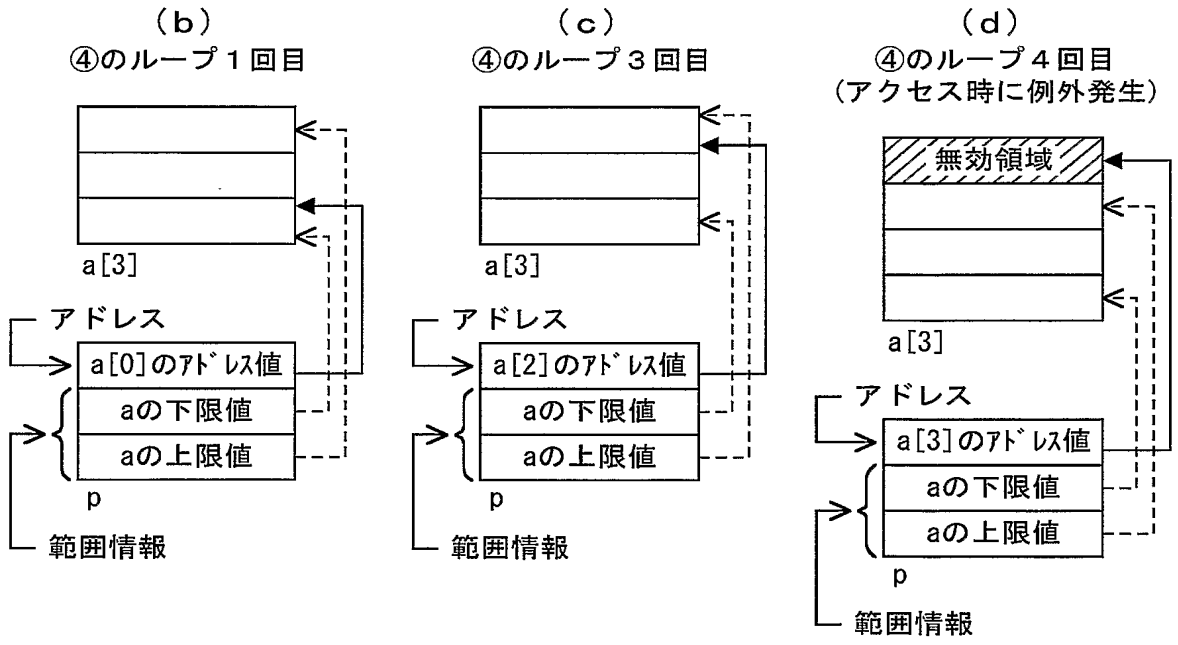
```
01: int main()
02: {
03:     int a[3];
04:
05:     foo(a);
06:
07:     return 0;
08: }
09:
10: void foo(int *p)
11: {
12:     int i;
13:
14:     for ( i = 0; i < 4; i++ )
15:         *p++ = 0;
16: }
```

① コンパイラがサイズ３を認識

② ①で認識したサイズから
範囲情報付きポインタを生成

③ 範囲情報付きポインタを受け取る

④ 不正なループ



【図３】

（ａ）

```
01: extern void input_str(char *str);
02:
03: int main()
04: {
05:     char buf[32];
06:     int  len = 0;
07:
08:     input_str(buf);
09:
10:     while ( buf[len] != 0)
11:         len++;
12:
13:     printf( "%d文字です¥n" , len);
14:
15:     return 0;
16: }
```

ソースコード(main.c)

（ｂ）

```
01: void input_str(char *str)
02: {
03:       int c;
04:
05:       while((c = getchar()) != EOF)
06:           *str++ = c;
07: }
```

ソースコード(input_int.c)

（ｃ）

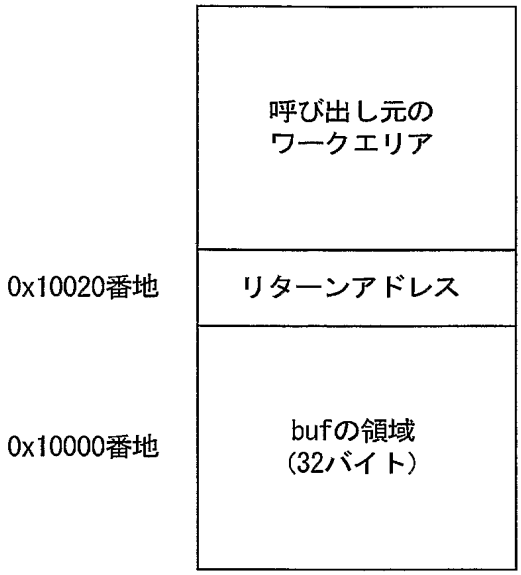
```
main.c:
      char[] UPPER          ;上方伸長
      *      FIXED          ;固定サイズ

input_str.c:
      *      FIXED          ;固定サイズ
```

障害復帰情報ファイル

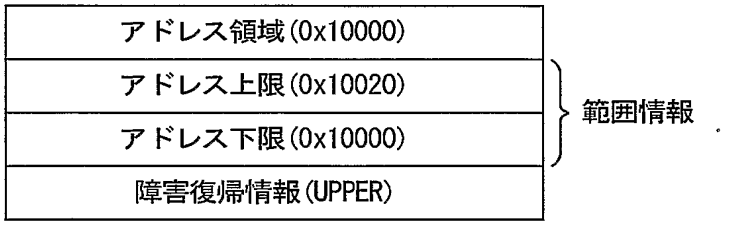
【図４】

（ａ）



スタックメモリ

（ｂ）



ポインタ値

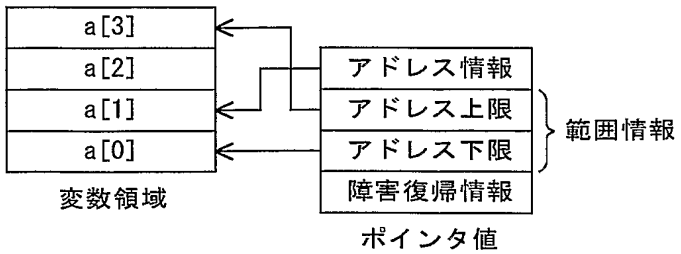
【図５】

（ａ）

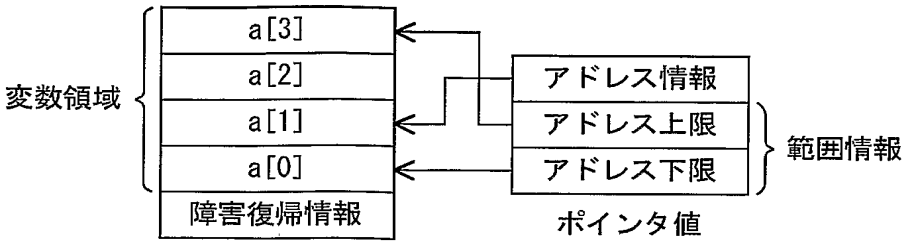
```
01: char a[5];
02:
03: extern void foo(char *p);
04:
05: int main()
06: {
07:       foo(&a[1]);
08:
09:       return 0;
10: }
```

Ｃ言語ソースコード

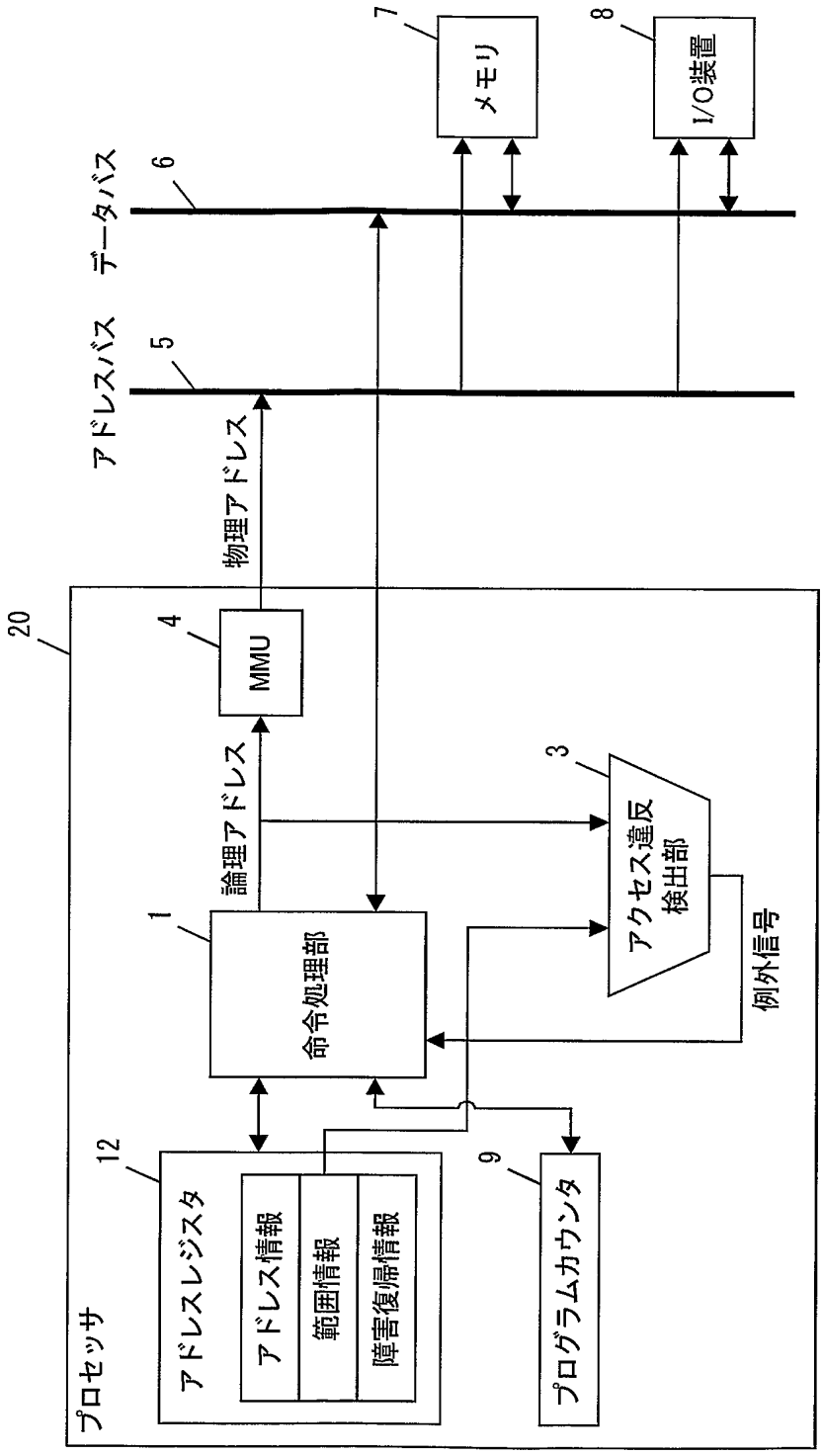
（ｂ）



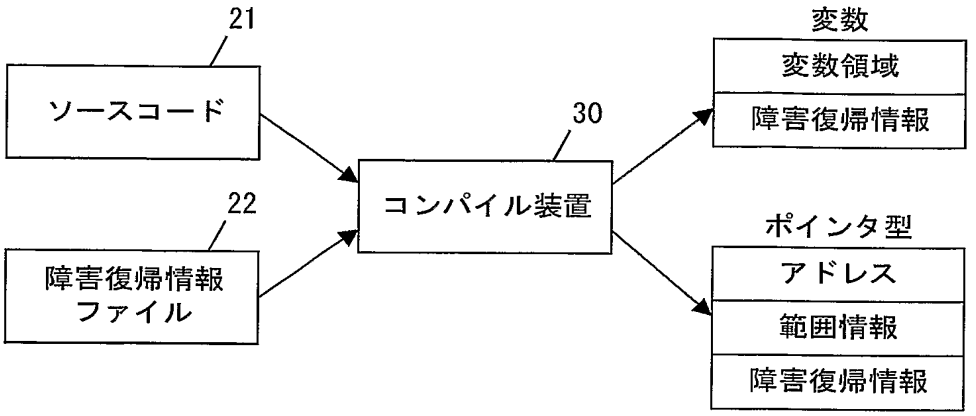
【図６】



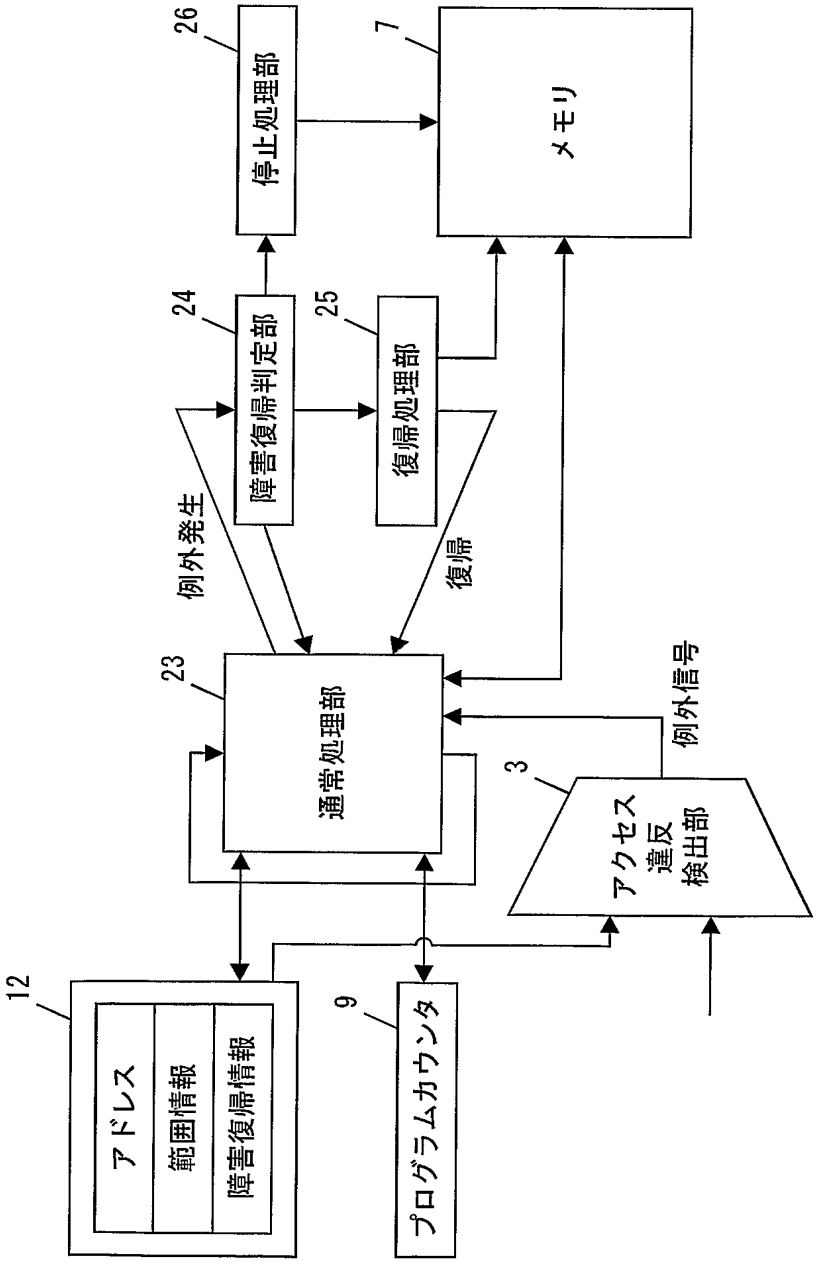
【図７】


7
メモリ
8
I/O装置
6
データバス
5
アドレスバス
物理アドレス
20
4
MMU
論理アドレス
3
アクセス違反
検出部
1
命令処理部
例外信号
12
アドレスレジスタ
アドレス情報
範囲情報
障害復帰情報
9
プログラムカウンタ
プロセッサ

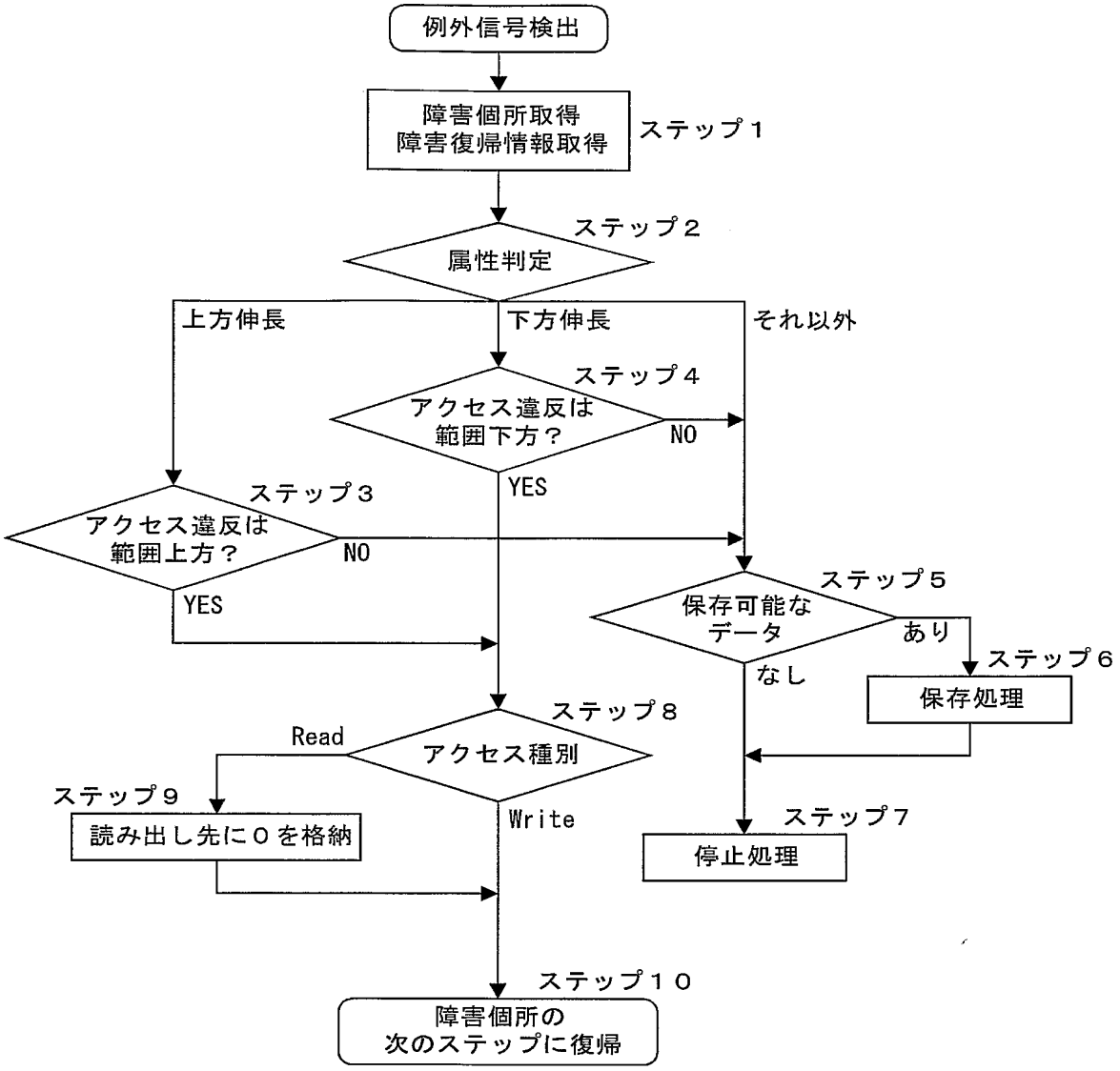
【図８】


21
ソースコード
22
障害復帰情報
ファイル
30
コンパイル装置
変数
変数領域
障害復帰情報
ポインタ型
アドレス
範囲情報
障害復帰情報

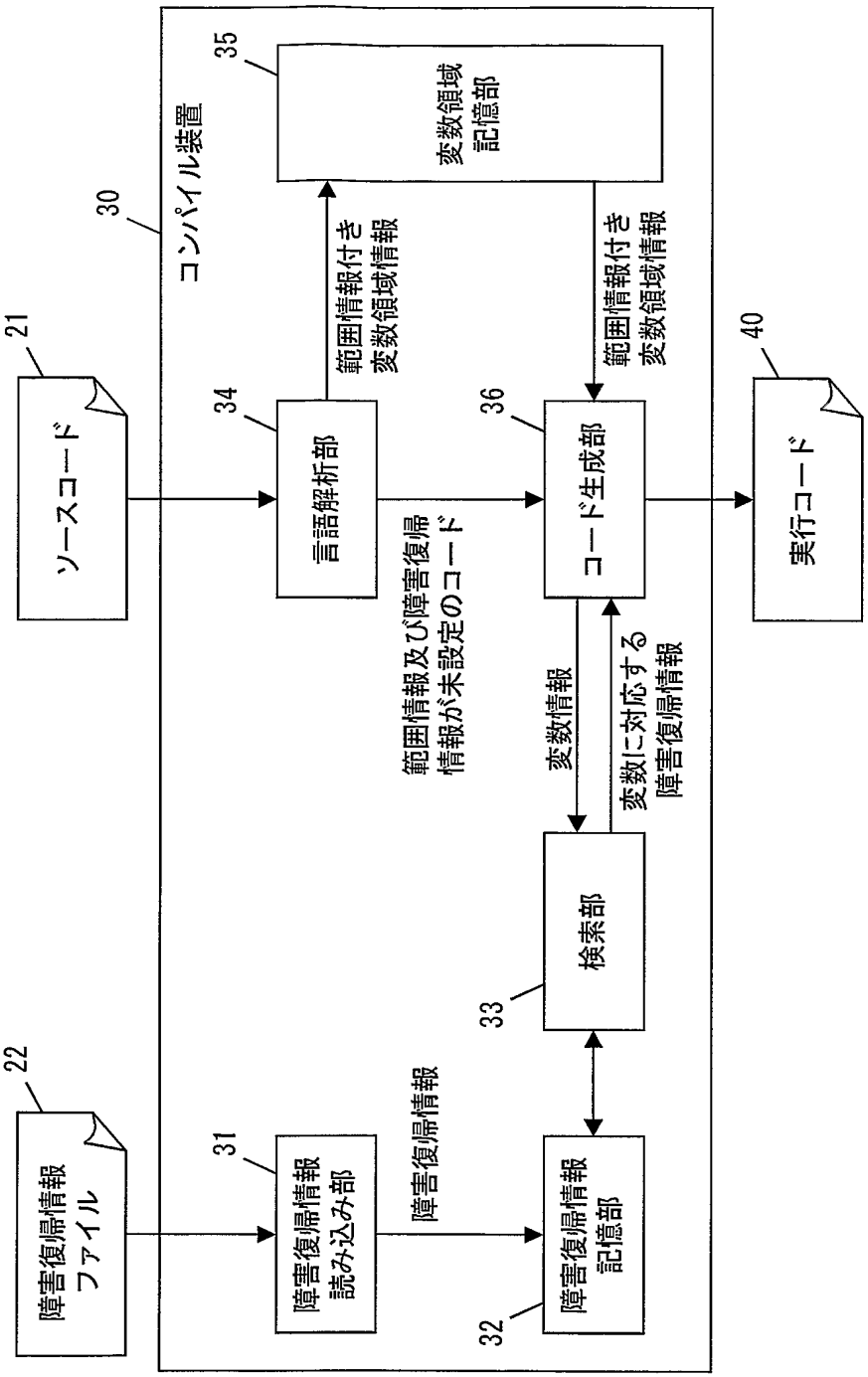
【図９】


26
停止処理部
7
メモリ
24
障害復帰判定部
25
復帰処理部
例外発生
復帰
23
通常処理部
例外信号
3
アクセス
違反
検出部
12
アドレス
範囲情報
障害復帰情報
9
プログラムカウンタ

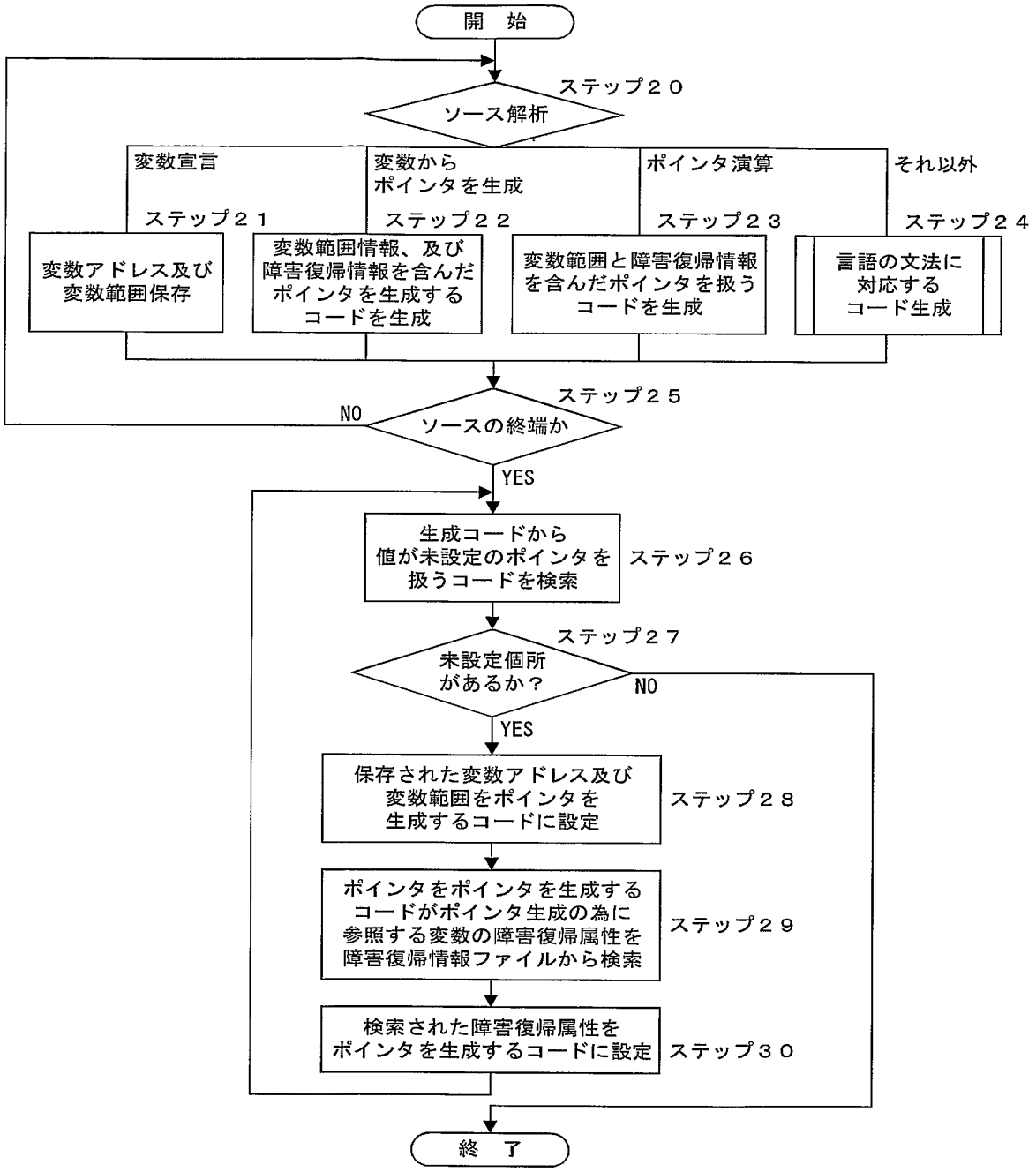
【図１０】


例外信号検出
障害個所取得
障害復帰情報取得
ステップ１
ステップ２
属性判定
上方伸長
下方伸長
それ以外
ステップ４
アクセス違反は
範囲下方？
NO
YES
ステップ３
アクセス違反は
範囲上方？
NO
YES
ステップ５
保存可能な
データ
あり
なし
ステップ６
保存処理
ステップ８
Read
アクセス種別
Write
ステップ９
読み出し先に０を格納
ステップ７
停止処理
ステップ１０
障害個所の
次のステップに復帰

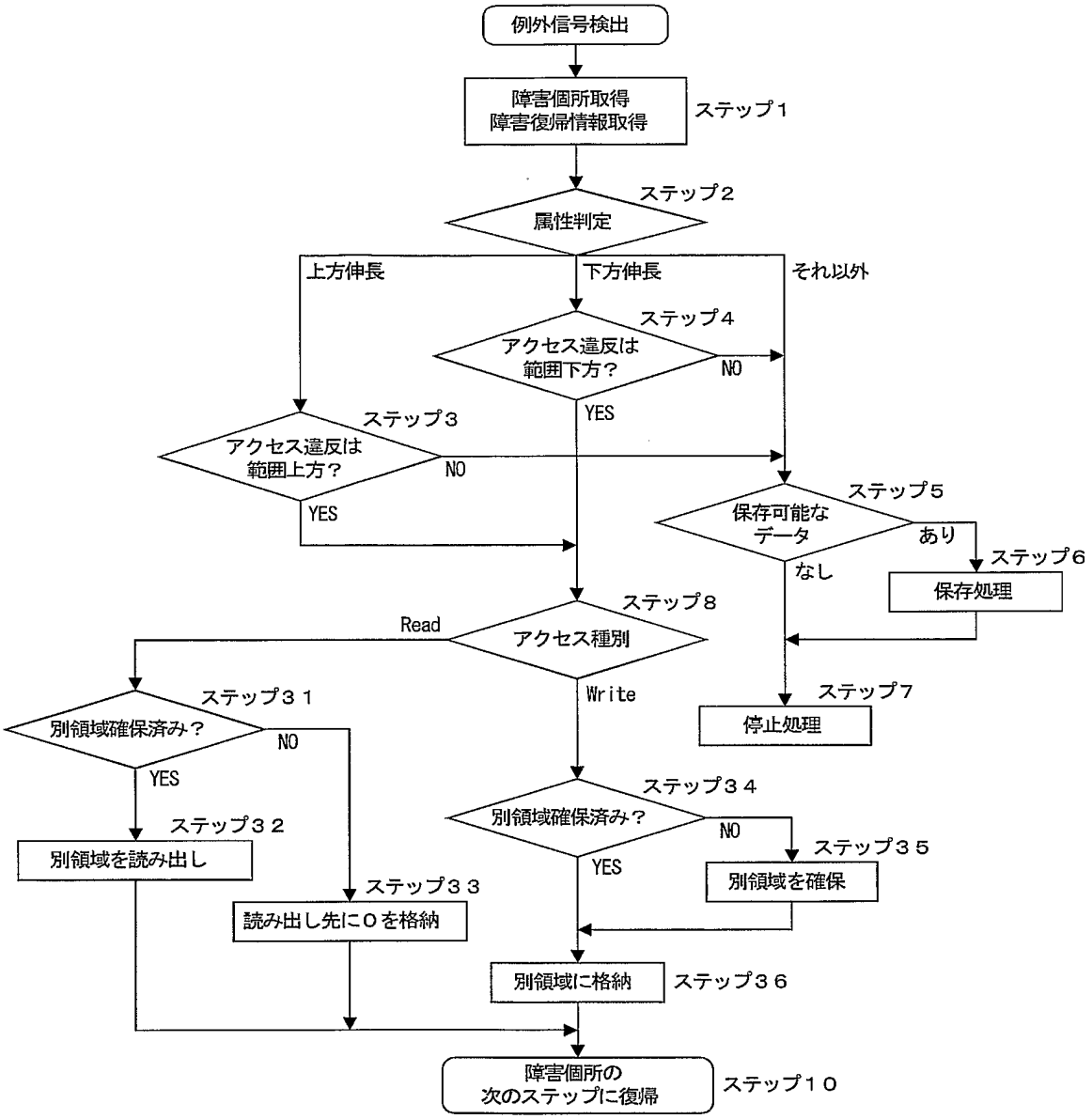
【図１１】


ソースコード
21
コンパイル装置
30
言語解析部
34
変数領域
記憶部
35
範囲情報付き
変数領域情報
範囲情報付き
変数領域情報
コード生成部
36
実行コード
40
範囲情報及び障害復帰
情報が未設定のコード
変数情報
変数に対応する
障害復帰情報
検索部
33
障害復帰情報
ファイル
22
障害復帰情報
読み込み部
31
障害復帰情報
障害復帰情報
記憶部
32

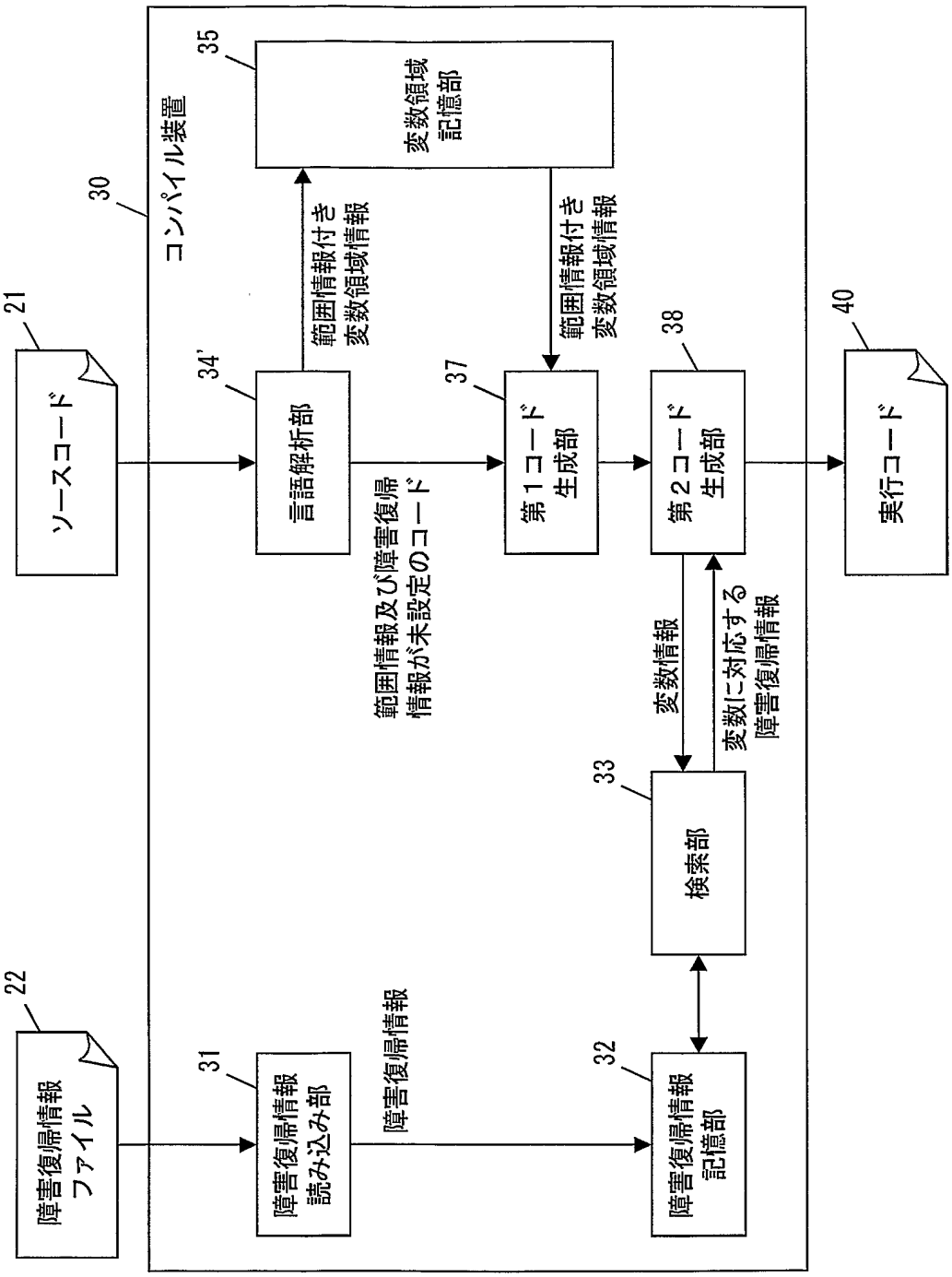
【図１２】


開　始
ステップ２０
ソース解析
変数宣言
変数から
ポインタを生成
ポインタ演算
それ以外
ステップ２１
ステップ２２
ステップ２３
ステップ２４
変数アドレス及び
変数範囲保存
変数範囲情報、及び
障害復帰情報を含んだ
ポインタを生成する
コードを生成
変数範囲と障害復帰情報
を含んだポインタを扱う
コードを生成
言語の文法に
対応する
コード生成
ステップ２５
NO
ソースの終端か
YES
生成コードから
値が未設定のポインタを
扱うコードを検索
ステップ２６
ステップ２７
未設定個所
があるか？
NO
YES
保存された変数アドレス及び
変数範囲をポインタを
生成するコードに設定
ステップ２８
ポインタをポインタを生成する
コードがポインタ生成の為に
参照する変数の障害復帰属性を
障害復帰情報ファイルから検索
ステップ２９
検索された障害復帰属性を
ポインタを生成するコードに設定
ステップ３０
終　了

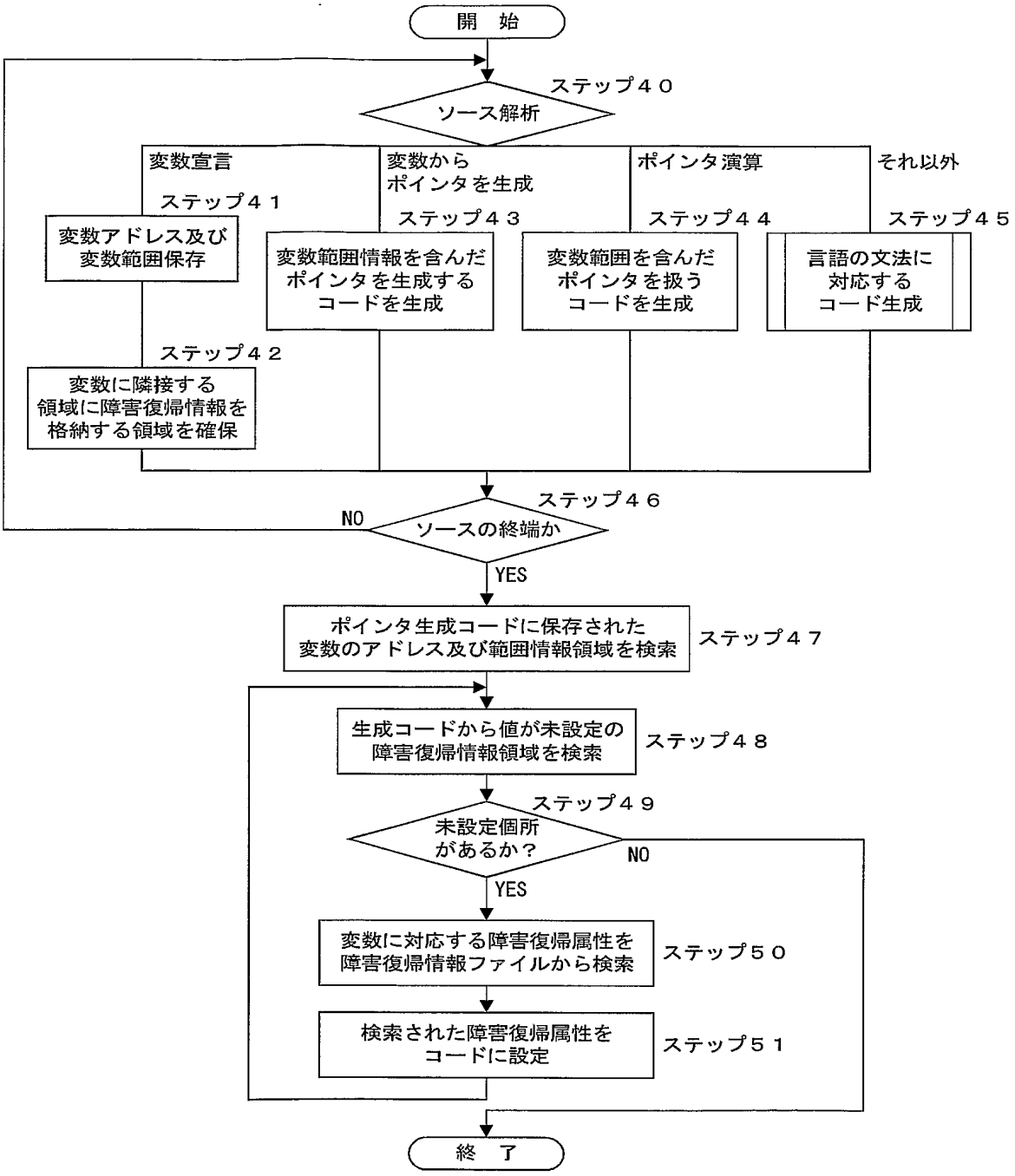
【図１３】


例外信号検出
障害個所取得
障害復帰情報取得
ステップ１
ステップ２
属性判定
上方伸長
下方伸長
それ以外
ステップ４
アクセス違反は
範囲下方？
NO
YES
ステップ３
アクセス違反は
範囲上方？
NO
YES
ステップ５
保存可能な
データ
あり
なし
ステップ６
保存処理
ステップ８
アクセス種別
Read
Write
ステップ７
停止処理
ステップ３１
別領域確保済み？
NO
YES
ステップ３２
別領域を読み出し
ステップ３３
読み出し先に０を格納
ステップ３４
別領域確保済み？
NO
YES
ステップ３５
別領域を確保
別領域に格納
ステップ３６
障害個所の
次のステップに復帰
ステップ１０

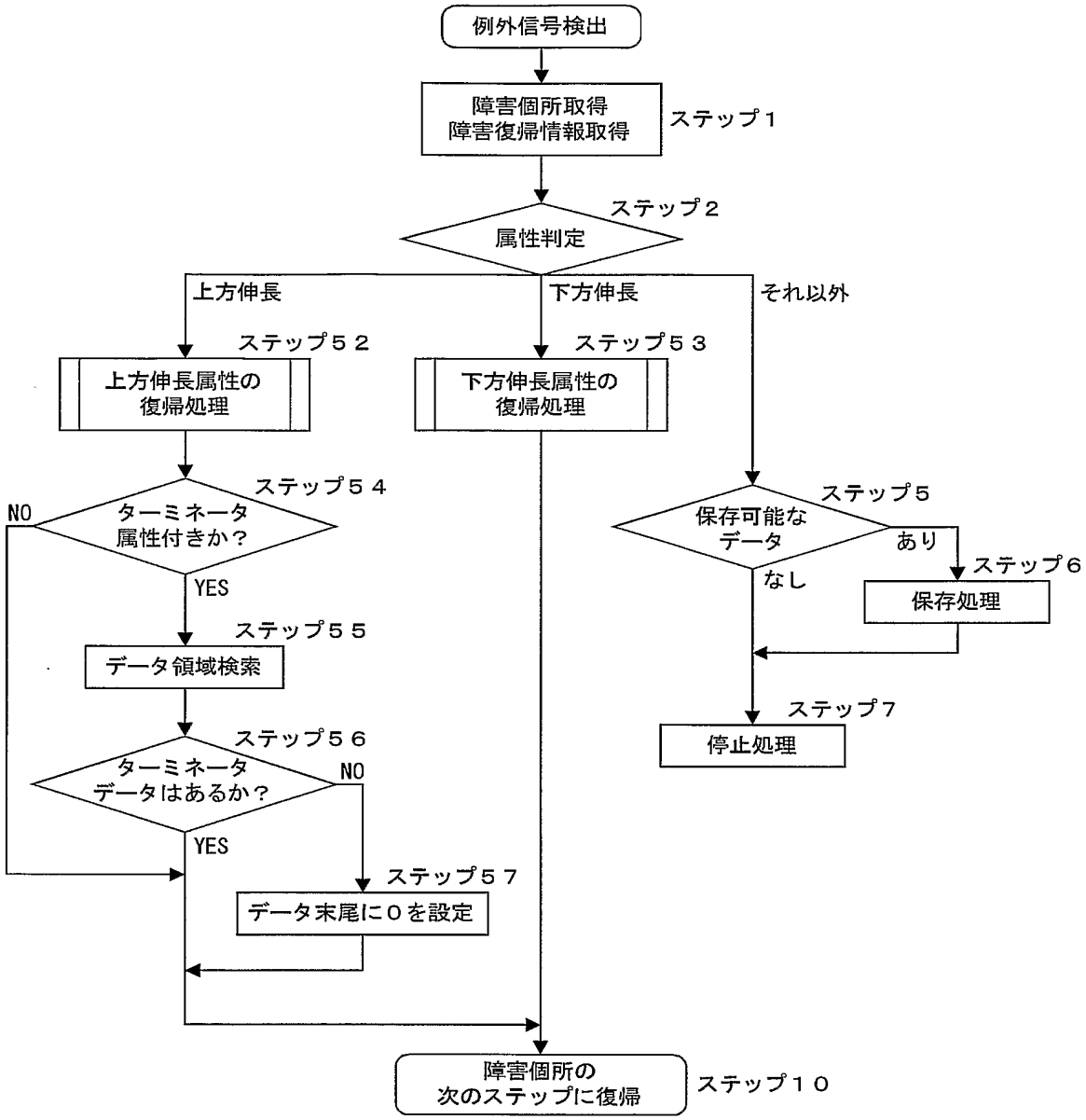
【図１４】


コンパイル装置
30
ソースコード
21
言語解析部
34'
変数領域
記憶部
35
範囲情報付き
変数領域情報
範囲情報付き
変数領域情報
第１コード
生成部
37
第２コード
生成部
38
実行コード
40
範囲情報及び障害復帰
情報が未設定のコード
変数情報
変数に対応する
障害復帰情報情報
検索部
33
障害復帰情報
ファイル
22
障害復帰情報
読み込み部
31
障害復帰情報
記憶部
32
障害復帰情報情報

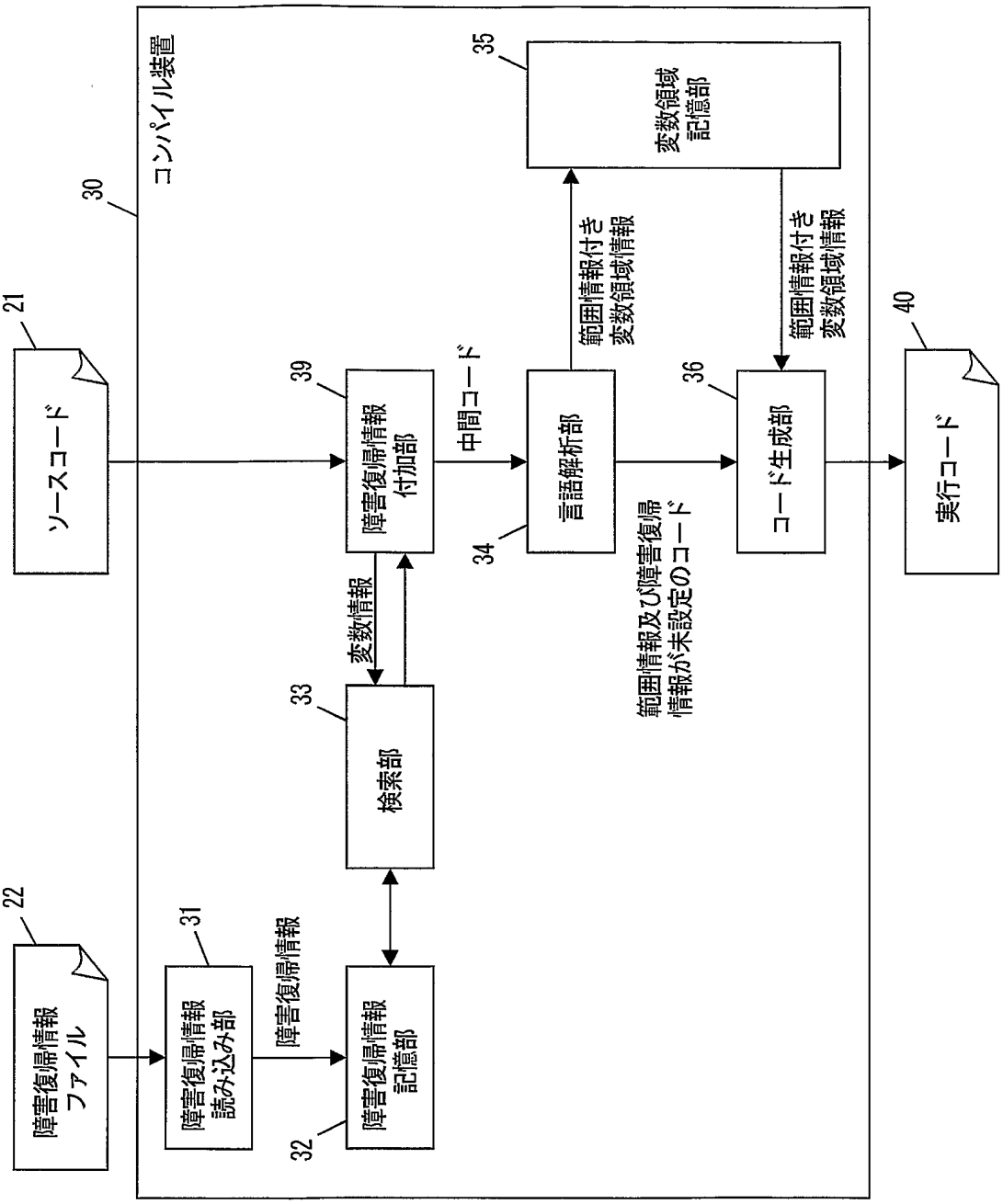
【図１５】


開　始
ステップ４０
ソース解析
変数宣言
変数から
ポインタを生成
ポインタ演算
それ以外
ステップ４１
変数アドレス及び
変数範囲保存
ステップ４３
変数範囲情報を含んだ
ポインタを生成する
コードを生成
ステップ４４
変数範囲を含んだ
ポインタを扱う
コードを生成
ステップ４５
言語の文法に
対応する
コード生成
ステップ４２
変数に隣接する
領域に障害復帰情報を
格納する領域を確保
ステップ４６
NO
ソースの終端か
YES
ポインタ生成コードに保存された
変数のアドレス及び範囲情報領域を検索
ステップ４７
生成コードから値が未設定の
障害復帰情報領域を検索
ステップ４８
ステップ４９
未設定個所
があるか？
NO
YES
変数に対応する障害復帰属性を
障害復帰情報ファイルから検索
ステップ５０
検索された障害復帰属性を
コードに設定
ステップ５１
終　了

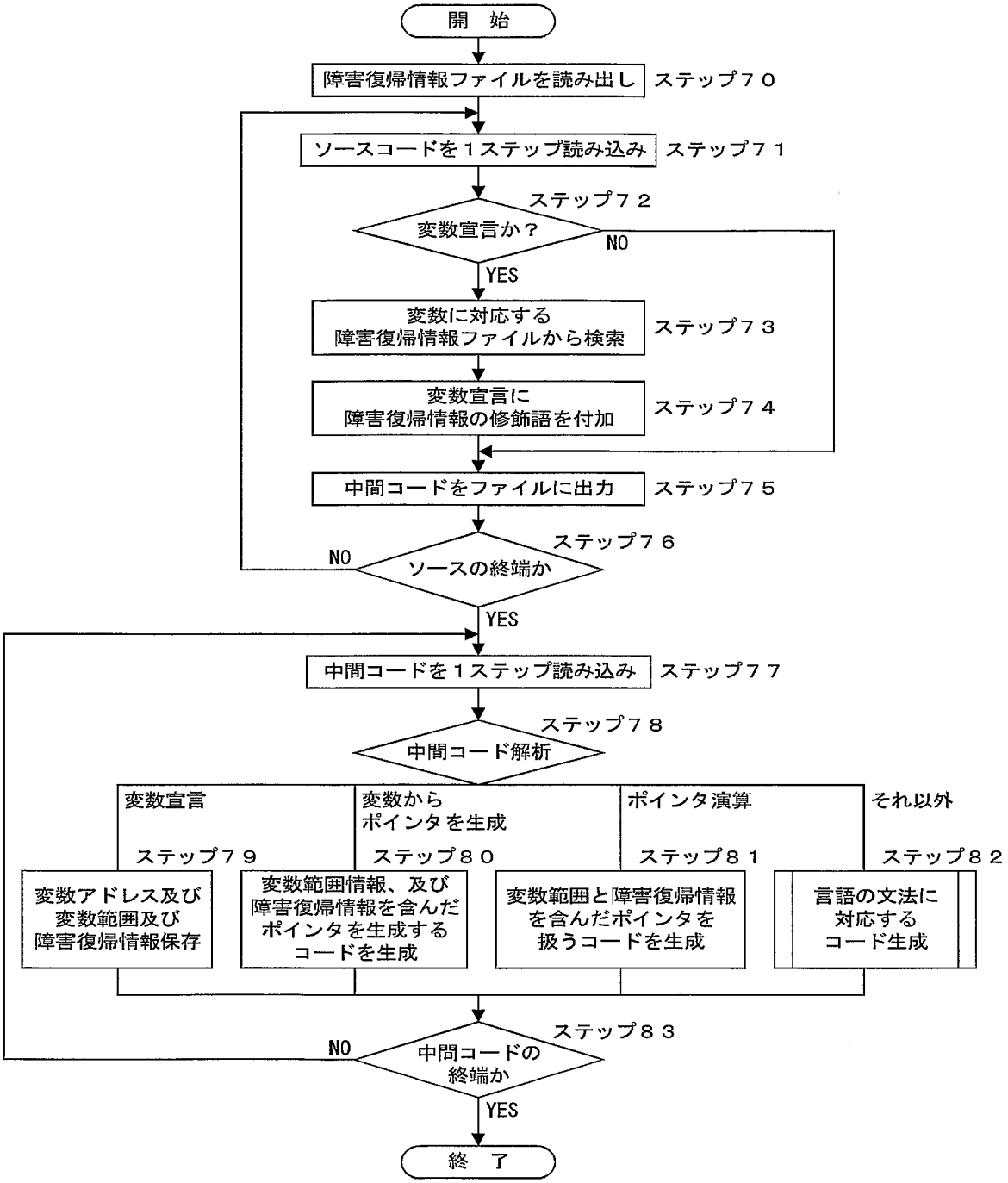
【図１６】


例外信号検出
障害個所取得
障害復帰情報取得
ステップ１
ステップ２
属性判定
上方伸長
下方伸長
それ以外
ステップ５２
上方伸長属性の
復帰処理
ステップ５３
下方伸長属性の
復帰処理
ステップ５４
NO
ターミネータ
属性付きか？
YES
ステップ５５
データ領域検索
ステップ５６
ターミネータ
データはあるか？
NO
YES
ステップ５７
データ末尾に０を設定
ステップ５
保存可能な
データ
あり
なし
ステップ６
保存処理
ステップ７
停止処理
障害個所の
次のステップに復帰
ステップ１０

【図１７】


コンパイル装置
30
ソースコード
21
障害復帰情報ファイル
22
障害復帰情報読み込み部
31
障害復帰情報記憶部
32
障害復帰情報
検索部
33
変数情報
障害復帰情報付加部
39
中間コード
言語解析部
34
範囲情報付き変数領域情報
変数領域記憶部
35
範囲情報及び障害復帰情報が未設定のコード
コード生成部
36
範囲情報付き変数領域情報
実行コード
40

【図１８】


開　始
障害復帰情報ファイルを読み出し
ステップ７０
ソースコードを１ステップ読み込み
ステップ７１
ステップ７２
変数宣言か？
NO
YES
変数に対応する
障害復帰情報ファイルから検索
ステップ７３
変数宣言に
障害復帰情報の修飾語を付加
ステップ７４
中間コードをファイルに出力
ステップ７５
ステップ７６
NO
ソースの終端か
YES
中間コードを１ステップ読み込み
ステップ７７
ステップ７８
中間コード解析
変数宣言
変数から
ポインタを生成
ポインタ演算
それ以外
ステップ７９
ステップ８０
ステップ８１
ステップ８２
変数アドレス及び
変数範囲及び
障害復帰情報保存
変数範囲情報、及び
障害復帰情報を含んだ
ポインタを生成する
コードを生成
変数範囲と障害復帰情報
を含んだポインタを
扱うコードを生成
言語の文法に
対応する
コード生成
ステップ８３
NO
中間コードの
終端か
YES
終　了

【図１９】

（ａ）

```
01: extern void input_str(char *str);
02:
03: int main()
04: {
05:     UPPER char buf[32];
06:     FIXED int  len=0;
07:
08:     input_str(buf);
09:
10:     while ( buf[len] ! = 0)
11:         len++;
12:
13:     printf( "%d文字です¥n" , len);
14:
15:     return 0;
16: }
```

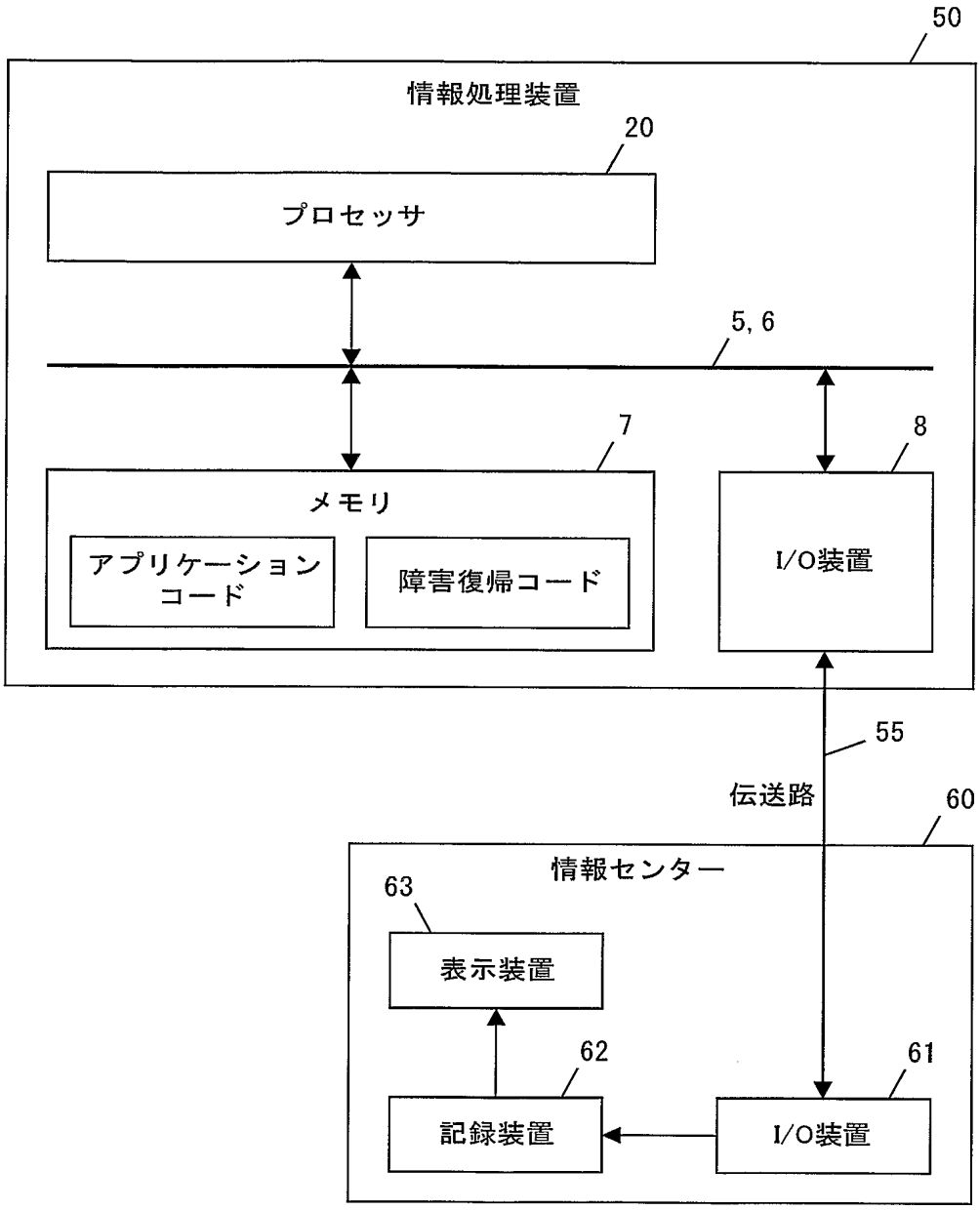
中間コード(main.cの処理後)

（ｂ）

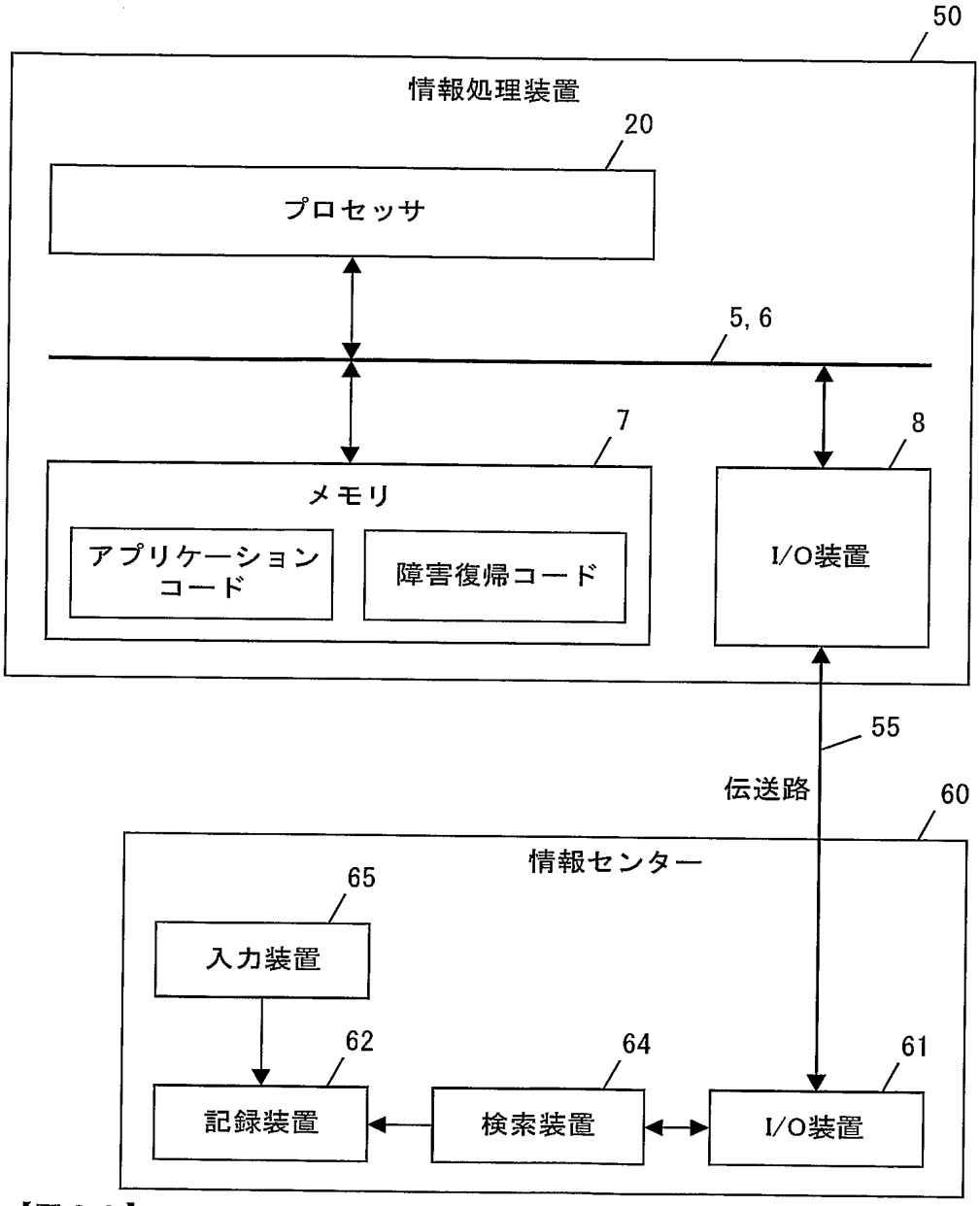
```
01: void input_str(char *str);
02: {
03:       FIXED int c;
04:
05:       while( (c = getchar()) ! = EOF)
06:           *str++=c;
07: }
```

中間コード(input_int.c)

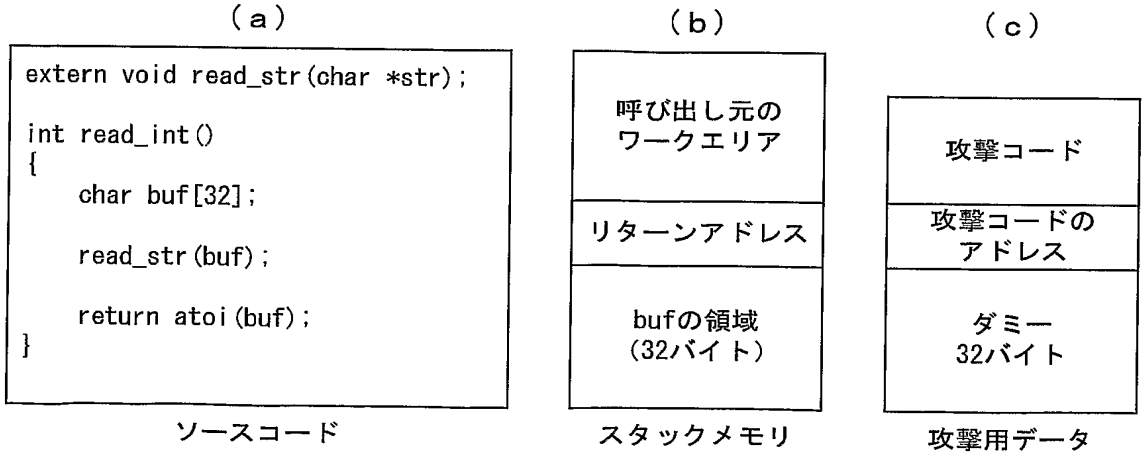
【図２０】


50
情報処理装置
20
プロセッサ
5, 6
7
8
メモリ
アプリケーション
コード
障害復帰コード
I/O装置
55
伝送路
60
情報センター
63
表示装置
62
61
記録装置
I/O装置

【図２１】


50
情報処理装置
20
プロセッサ
5, 6
7
8
メモリ
アプリケーション
コード
障害復帰コード
I/O装置
55
伝送路
60
情報センター
65
入力装置
62
64
61
記録装置
検索装置
I/O装置

【図２２】


（ａ）
（ｂ）
（ｃ）
extern void read_str(char *str);
int read_int()
{
char buf[32];
read_str(buf);
return atoi(buf);
}
呼び出し元の
ワークエリア
リターンアドレス
bufの領域
（32バイト）
攻撃コード
攻撃コードの
アドレス
ダミー
32バイト
ソースコード
スタックメモリ
攻撃用データ

【書類名】要約書
【要約】
【課題】　情報処理装置の障害復帰機能を強化する。
【解決手段】　ソースコードのコンパイルの段階で、ポインタ変数に、ポインタの指す変数の範囲情報と、障害復帰用の情報をもたせておく。範囲情報を元にポインタアクセスの不正を検出した際に、障害復帰情報も取得できるので、障害復帰情報に従って、傷害の原因になったデータに適した復帰処理を実施する。範囲情報と障害復帰情報とを、アドレスに関連付けることにより、障害復帰情報（ＵＰＰＥＲ、ＬＯＷＥＲ、ＦＩＸＥＤ）を用いて、障害個所のメモリ領域のデータタイプを特定できる。
【選択図】　図９

特願２００４－００８３２７

出　願　人　履　歴　情　報

識別番号　　[０００００５８２１]

１．変更年月日　　１９９０年　８月２８日
[変更理由]　　新規登録
住　所　　大阪府門真市大字門真１００６番地
氏　名　　松下電器産業株式会社